TREBALL DE FI DE CARRERA

TÍTOL: Detector de limitacions en piles TCP/IP

AUTOR: Alberto José Juan Torres

DIRECTOR: Lluís Casals Ibáñez

DATA: 7 de juliol de 2006

Títol: Detector de limitacions en piles TCP/IP

Autor: Alberto José Juan Torres

Director: Lluís Casals Ibáñez

Data: 7 de juliol de 2006

Resum

Aquest treball de fi de carrera pretén ser un estudi sobre les limitacions en la implementació de la pila TCP/IP sobre dispositius que disposen de pocs recursos, ja que la tendència dels nous dispositius és incloure el màxim possible de característiques en aparells més petits. Aquestes característiques inclouen la comunicació amb una xarxa i l'intercanvi de dades.

Aquest estudi analitza les limitacions més habituals de la implementació de la pila TCP/IP i es donen a conèixer les proves que s'haurien de fer sobre dispositius limitats per tal de determinar les seves capacitats i poder mantenir una comunicació fiable. Aquestes proves s'expliquen pas a pas, donant les pautes teòriques que s'han seguit en el seu desenvolupament

Per acabar, també s'ha implementat una aplicació que realitza part d'aquestes proves descrites i permet donar una idea de les característiques d'un dispositiu connectat a una xarxa.

Title: TCP/IP stack limits detector

Author: Alberto José Juan Torres

Director: Lluís Casals Ibáñez

Date: July, 7th 2006

Overview

This final career work pretends to be a study about the TCP/IP stack limitations, when the stack is implemented over low resources devices. Nowadays, there's going a trend towards designing smaller devices, but people want the maximum number of features on these devices, and communications are included in that features. The problem is: this communications need protocols, and protocols need a protocol stack.

This study talks about the analysis of the limitations that affect TCP/IP stack and explain in great detail all the tests we should do over a limited device, to get conclusions about the capacities of these devices if they are connected to a network.

Finally, the last part describes the creation of an application that implements part of the tests explained previously and gives an idea about the features of the limited device.

ÍNDEX

INTRODUCCIÓ

Actualment, quan parlem de tecnologia, quant més petits són els dispositius, millor. Però no ens val amb tenir dispositius minúsculs, sinó que també volem que aquest aparells realitzin el major número de funcions possible, i bona part d'aquestes funcions estan relacionades amb les comunicacions.

Aquestes comunicacions necessiten seguir una sèrie de requisits per a portar-se a terme correctament. Aquests requisits venen descrits als protocols, però amb això no basta, un únic protocol no es sap ocupar d'una comunicació completa, es necessiten distints protocols, cadascun dedicat a unes tasques determinades. El conjunt d'aquests protocols s'organitza com una pila de protocols.

La pila de protocols s'ha d'aconseguir implementar sobre dispositius petits, que tenen capacitat de memòria i de processament menors que les que necessita la pila. Llavors el que s'ha de fer és treure capacitats a aquesta pila per tal de poder incloure-la al dispositiu i que aquest pugui establir comunicacions.

L'estudi i l'aplicació que es realitza en aquest projecte es pot situar en diferents àmbits, sempre que existeixi un conjunt de màquines connectades en xarxa. En nostre cas es pretén explicar una d'aquestes possibles situacions en les que seria útil l'aplicació proposada.

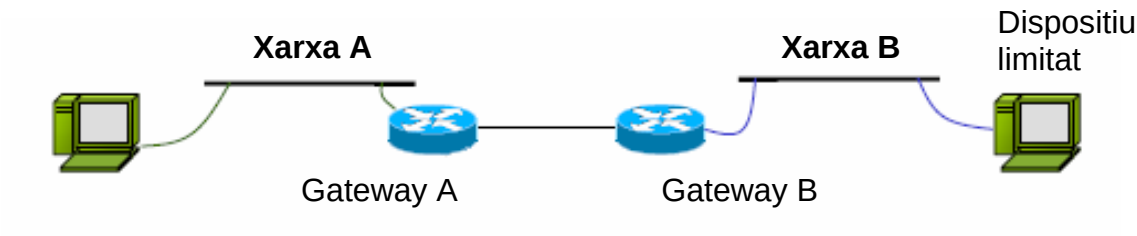

Fig I.1 Entorn d'aplicació

Imaginem que es disposa de dues xarxes locals, que mitjançant dos *Gateways* estan connectades entre si. Totes les màquines d'aquestes dues xarxes utilitzen piles TCP/IP. Una d'aquestes xarxes (l'anomenarem xarxa A) està formada per dispositius d'alt nivell, amb multitud de recursos disponibles per al processament de paquets i contenen cadascun d'ells una implementació completa de la pila, sense cap tipus de restricció. Per l'altra banda, la segona xarxa (xarxa B) està formada per màquines que disposen de limitacions de processament i de memòria, i que implementen piles TCP/IP adaptades als seus poc recursos, per tant, només tenen l'imprescindible.

Si una màquina de la xarxa B vol enviar paquets a una de la xarxa A, llavors no hi haurà cap tipus de problema, ja que la màquina de "A" podrà reconèixer el paquet que s'ha enviat. On realment comencen els dubtes, és quan la comunicació va en sentit contrari, és a dir de xarxa "A" a xarxa "B", ja que és molt possible que un paquet d'un dispositiu "A" no pugui ser interpretat per el

dispositiu “B” degut a que contingui algun tipus de paràmetre que la pila limitada desconegui.

La solució a aquest problema seria la implementació al *Gateway* d’entrada a la xarxa “B” d’una aplicació que controlés quin tipus de limitacions té cada dispositiu de B, d’aquesta manera, podria adaptar els paquets que venen de la xarxa “A” a les limitacions de les màquines de la xarxa B. El nostre estudi es centra en part de la detecció de limitacions a la pila, no entrem al procés d’adaptar els paquets per a que puguin ser reconeguts per la pila destí.

Aquest TFC vol donar a conèixer aquestes limitacions que afecten a la implementació de la pila TCP/IP, i per això es realitza un estudi sobre les possibles limitacions en la implementació dels protocols quan s’adapten a dispositius de pocs recursos. També té com a objectiu crear una aplicació que ens permeti realitzar una sèrie de proves per a determinar en quin grau es troba limitada una màquina connectada a la nostra xarxa.

Al primer capítol s’expliquen les piles de protocols, i es donen dos exemples, el Model OSI i la pila TCP/IP. Tot seguit es passa a descriure els protocols de la pila TCP/IP que s’utilitzen en aquest TFC.

Al capítol 2, es parla de les limitacions en la implementació de la pila TCP/IP, i després de fer una introducció a piles limitades que s’han aconseguit desenvolupar, es procedeix a descriure les proves que s’haurien de fer per a determinar aquestes limitacions, en aquest cas sobre el protocol IP i TCP.

El tercer capítol està completament dedicat a l’aplicació que s’ha implementat. Per començar es fa una descripció de l’aplicació en sí, i es donen instruccions per al seu correcte funcionament. Tot seguit, s’expliquen les proves implementades i el procediment que segueixen per a arribar a deduir en quin grau està limitada la pila.

El quart i últim capítol conté les conclusions que s’han obtingut a partir de la realització de l’estudi i, sobretot, de l’aplicació, el que més temps s’ha dedicat en la realització d’aquest projecte.

CAPÍTOL 1. FONAMENTS TEÓRICS

1.1. Piles de protocols

Aquest primer capítol està dedicat a explicar la base d'aquest projecte, les piles de protocols (*protocol stacks*), mes concretament, la pila TCP/IP, que és la més estesa i en la que ens hem centrat en el nostre estudi.

1.1.1. Què és una pila de protocols?

Es defineix com a pila de protocols a les arquitectures de protocols que estan organitzades en diferents capes, cadascuna lligada a una tasca determinada. A cada capa de la pila s'utilitza un protocol que regeix la comunicació amb el nivell parell de la pila de la màquina destí. Per tant, cada capa de la pila té la funció de proporcionar serveis a la capa immediatament superior.

Per tant, podríem dir que la pila és una jerarquia de protocols, distribuïts en capes, que comença al nivell d'aplicació, on es comunica amb la font de les dades a transmetre, va passant per cadascuna de les capes fins que arriba al nivell físic, que envia les dades pel mitjà corresponent (cable, aire, etc.).

1.1.2. Model OSI

Amb la finalitat de realitzar un model de referència de piles de protocols, la ISO (*International Standards Organization*) va publicar l'any 1984 el Model OSI (*Open Systems Inerconnect*), on es definien les diferents capes que hauria de tenir una pila de protocols. Es definia així el primer pas cap a un estàndard en quant a l'organització de protocols per a les comunicacions en xarxa.

Els objectius d'aquest model eren:

- Establir un estàndard que permetés la comunicació entre dispositius de diferents fabricants
- Definir les funcions específiques de cada nivell de la pila, així com donar una guia per a la implementació de la comunicació entre capes
- Detallar el conjunt de protocols que s'hauria d'utilitzar a cada capa

Un dels punts a destacar de la pila OSI és que cada nivell actua com una unitat independent. Això vol dir que (teòricament), es podria substituir el protocol que s'utilitza a una certa capa per un altre protocol sense que els nivells superior e inferior a aquesta capa en surtin afectats.

Fig 1.1 Model OSI

El model OSI ve definit per 7 nivells:

- *Aplicació*: Interfície que comunica les fonts i destins de dades amb el sistema de comunicació.
- *Presentació*: Adapta el format de les dades i la seva codificació per a que es pugui transmetre entre sistemes diferents.
- *Sessió*: Gestiona el diàleg entre dispositius (inici, final, etc.)
- *Transport*: Controla la transmissió extrem a extrem, gestiona els errors, controla el flux de dades, etc.
- *Xarxa*: Estableix la ruta a seguir per els paquets
- *Enllaç*: S'encarrega del lliurament de dades entre nodes d'un enllaç de xarxa. Assegura la fiabilitat del mitjà de transmissió, realitza controls d'errors, retransmissions, etc.
- *Físic*: S'encarrega de passar el bits al mitjà de transmissió (cable de coure, aeri, fibra, etc.)

Aquest model es va realitzar, com ja s'ha comentat, per a definir un estàndard en quant a piles de protocols, però en el moment de la seva creació, ja existien altres piles molt utilitzades, com TCP/IP.

1.1.3. Pila TCP/IP

Encara que el model OSI sigui l'estàndard, abans que aquest model sortís a la llum, es va desenvolupar una família de protocols que ja realitzava les operacions descrites al model OSI. Aquesta família era la família de protocols TCP/IP, que va començar a desenvolupar-se a principis dels anys 70. Actualment és la més estesa, sobretot degut a que és la que s'utilitza a Internet.

El nostre estudi estarà dedicat exclusivament a aquesta pila, més en concret ens dedicarem als protocols de la capa de transport (TCP) i de la capa de xarxa (IP).

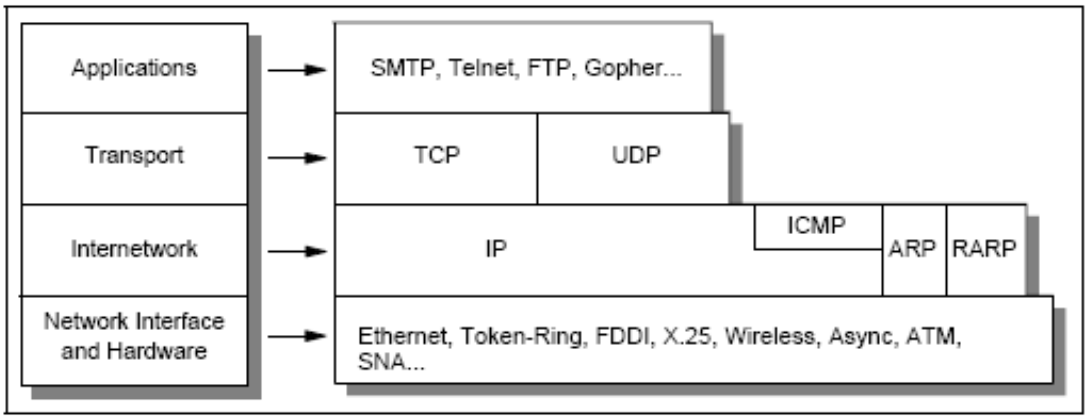

Fig 1.2 Pila TCP/IP i protocols que utilitza

A diferencia d'OSI, la pila TCP/IP només té quatre capes:

- *Network Interface and Hardware Layer (Nivell d'enllaç)*: Aquesta capa, també anomenada *Link-Layer*, especifica com són transportades les dades a través dels mitjans físics (cable, aire, etc.). Un exemple de protocol d'enllaç pot ser Ethernet, un dels més utilitzats a xarxes d'àrea local.

- *Internetwork Layer (Nivell de xarxa)*: Aquesta és una de les capes més importants de la pila, ja que és la que permet enviar dades entre un origen i un destí. També coneguda com *Internet Layer*, té com a protocol principal IP.

- *Transport Layer (Nivell de transport)*: Els protocols d'aquesta capa estan orientats a donar fiabilitat a la capa superior i assegurar que les dades arriben correctament. El protocol més conegut és el TCP, que permet realitzar una transmissió de dades totalment fiable, i que serà part essencial en aquest TFC.

- *Application Layer (Nivell d'aplicació)*: Com en el cas del model OSI, aquest nivell conté els protocols necessaris per a la comunicació amb programes que utilitzin la pila per a enviar o rebre dades. Aquests protocols poden ser FTP, HTTP o Telnet, entre altres. Són els encarregats de passar les dades que arriben de la capa de transport a un format conegut per l'aplicació que espera les dades. Un clar exemple seria un navegador de Internet (aplicació), que espera paquets que viatgin a sobre del protocol HTTP (protocol de la capa d'aplicació).

Com ja hem dit, la pila TCP/IP realitza les mateixes operacions que el model OSI, però TCP només te quatre capes i OSI en té set. Això vol dir que hi ha capes de TCP/IP que realitzen les funcions de dues o més del model OSI.

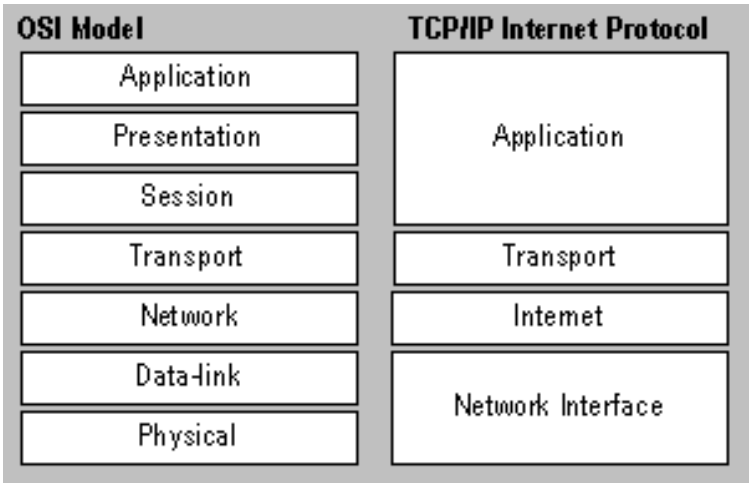

Fig 1.3 Comparació model OSI i pila TCP/IP

A la figura 1.3 podem observar la distribució de la pila TCP/IP respecte el model OSI. Veiem que la capa d'aplicació TCP/IP conté les capes d'aplicació, de presentació i de sessió d'OSI. Això és degut a que els protocols que implementa aquesta capa de TCP estan dissenyats per a poder realitzar totes aquestes funcions, sense tenir que utilitzar un protocol a cada capa. A continuació veiem que les capes de transport i xarxa són equivalents en els dos models, i per acabar, que el nivell de interfície de xarxa de la pila TCP/IP inclou el nivell d'enllaç i el nivell físic de OSI.

1.2. Descripció dels protocols estudiats

Aquest apartat no pretén ser una explicació completa dels RFC (*Request for Comments*) de cadascun dels protocols, sinó que el seu objectiu és donar uns mínims coneixements relacionats amb els protocols que s'utilitzen tant a l'aplicació com a les proves teòriques. Per tant, només ens centrarem en les parts que afecten directament a aquest TFC, deixant de banda les que no són imprescindibles.

1.2.1. IP

El Protocol de Internet (*Internet Protocol*) es va dissenyar per a l'intercanvi de bloc de dades entre màquines connectades a una xarxa de comunicacions [2]. Aquest protocol proporciona tots els mitjans necessaris per a enviar aquests blocs de dades, també anomenats paquets o datagrames, des de un origen fins

a un destí. Aquest origen i destí venen definits per una adreça, que permet identificar-los a dins de la xarxa.

Cada paquet que s'envia, conté dues parts diferenciades, la capçalera i les dades. La capçalera conté la informació necessària per a que el paquet arribi al seu destí. Al camp de dades, s'envia la informació que es vol transmetre al destí. A continuació, a la figura 1.3, veiem la definició de la capçalera IP

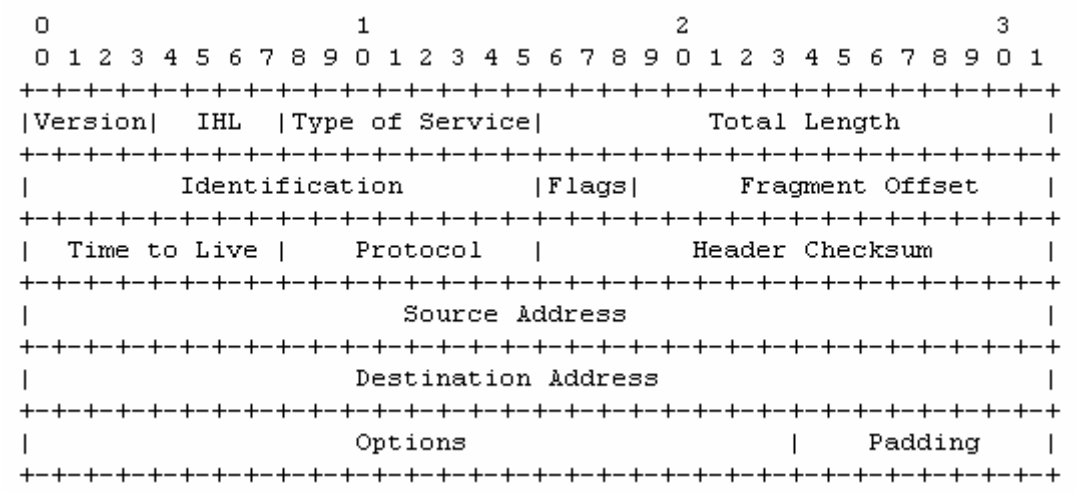

Fig 1.4 Capçalera IP

Es pot observar que, a més de les adreces origen i destí, la capçalera també inclou altres dades interessants per a la transmissió:

- *Version*: Indica la versió del protocol IP. Actualment és la IPv4, però en un futur no molt llunyà, es passarà a la IPv6, que permet un nombre molt major d'adreces, entre altres coses.
- *IHL (Internet Header Lenght)*: Dóna el valor de la longitud de la capçalera, com a mínim ha de ser de 20 bytes, però pot ser major si conté opcions.
- *Type of Service*: Proporciona uns certs valors per a indicar la qualitat de servei (QoS) que s'ha de proporcionar a aquell paquet (prioritari, urgent, rutina, etc.).
- *Identification (ID)*: Dóna un identificador al paquet, principalment per ajudar a ajuntar paquets en una possible fragmentació.
- *Flags*: S'utilitza únicament a la fragmentació com a indicador de control.
- *Fragment Offset*:També s'utilitza exclusivament a paquets fragmentats. Indica a quina part del datagrama original pertany el fragment.
- *Time to Live (TTL)*: Conté un valor que indica el nombre màxim de dispositius que pot travessar aquest paquet durant el seu viatge. Cada un d'ells disminueix en 1 el valor del TTL i quan arriba a 0, el paquet s'elimina.

- *Protocol*: Diu de quin protocol de nivell superior són les dades que es transporten (TCP, UDP, etc.).
- *Header Checksum*: Control d'errors de la capçalera (no del camp de dades).
- *Source Address*: Adreça origen.
- *Destination address*: Adreça destí.
- *Options i Padding*: Camp que conté dades opcionals del protocol. Si l'opció no ocupa un múltiple de 32 bits, llavors s'han d'afegir bits de farciment (padding) fins a completar la paraula de 32 bits.

Tot i això, IP únicament és capaç de proporcionar un servei *best-effort* (fer el millor possible) en la transmissió de datagrames, que no és fiable, ja que no assegura que el paquet arribi al seu destí, en l'ordre correcte o que el paquet no estigui malmès (el checksum només es calcula per a la capçalera). Si es vol aquesta fiabilitat, s'ha d'acudir a protocols de nivell superior, com per exemple, TCP.

1.2.2. ICMP

El Protocol de Control de Missatges d'Internet (*Internet Control Message Protocol*) està destinat a enviar missatges generats, normalment, en resposta a un error de paquets IP. Aquest error pot ser degut, per exemple, a que el destí és inaccessible, o que el TTL d'un datagrama ha arribat a 0.

ICMP no té com a objectiu donar fiabilitat a IP, sinó que pretén subministrar informació sobre els problemes que existeixen a la xarxa. Els missatges ICMP viatgen a dins de paquets IP, tot i així, IP i ICMP es consideren protocols del nivell de xarxa. El fet de que ICMP vagi sobre IP pot ocasionar que un missatge d'error no arribi al seu destí, degut a la no fiabilitat de IP.

1.2.2.1. ICMP ECHO

Podem trobar diferents paquets ICMP, cadascun destinat a un tipus d'error específic, només hi ha dos camps de la capçalera ICMP que no varia, el *Type* i el *Code* mitjançant els quals es sap quin error és i quin format de paquet s'ha d'utilitzar.

No entrarem a explicar tots els tipus de paquet, sinó que ens centrarem en el que utilitzem per a la nostra aplicació, el ECHO REQUEST i el ECHO REPLY. Tots dos formen part d'un diàleg que és l'utilitzat per l'eina PING amb l'objectiu de comprovar si una certa màquina es troba disponible. El procediment és enviar un missatge ICMP ECHO REQUEST i esperar una resposta del tipus ICMP ECHO REPLY.

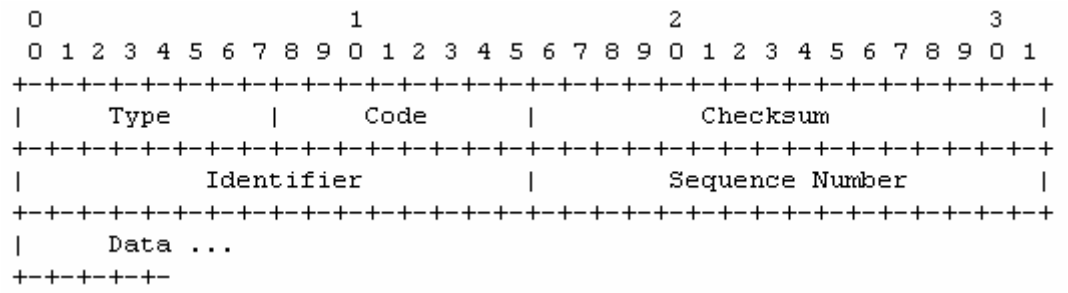

Fig 1.5 Capçalera ICMP ECHO

Com ja s'ha comentat, les capçaleres canvien per a cada tipus de missatge, en el cas del ECHO, el REQUEST i el REPLY comparteixen la mateixa:

- *Type*: Indica el tipus de paquet ICMP
- *Code*: No s'utilitza al ECHO.
- *Checksum*: Control d'errors
- *Identifier*: ID del paquet, el REPLY ha de portar el mateix ID que el REQUEST per a identificar-lo.
- *Sequence Number*: Al igual que l'*Identifier*, REQUEST i REPLY han de portar el mateix número de seqüència.

Com a últim apunt, segons el RFC 1122 (veure [5]), on es mostren els requeriments d'un host de Internet, tota màquina connectada a una xarxa ha d'implementar el ICMP ECHO.

1.2.3. TCP

El Protocol de Control de Transmissió (*Transmission Control Protocol*) [3] es troba al nivell de transport de la pila, un per damunt d'IP, i té com a principal objectiu oferir una transmissió de dades fiable que no s'obté amb IP. TCP és un protocol orientat a connexió, que assegura una transmissió ordenada, lliure d'errors, sense pèrdues de paquets i sense duplicacions.

La filosofia que segueix TCP és que la transmissió ha de ser totalment fiable, i això s'aconsegueix amb missatges de confirmació. Un cop la màquina destí rep un paquet, el que es fa és generar un missatge ACK i l'envia a l'origen. El que s'aconsegueix amb això és fer saber a la màquina que transmet dades, que aquestes han arribat correctament i que pot seguir enviant. En el cas de no arribar aquesta confirmació, es realitza una retransmissió del paquet.

Els paquets TCP estan formats per una capçalera i un camp de dades, i aquests viatgen dins el camp de dades d'IP. A la figura 1.5 veiem com està formada la capçalera TCP.

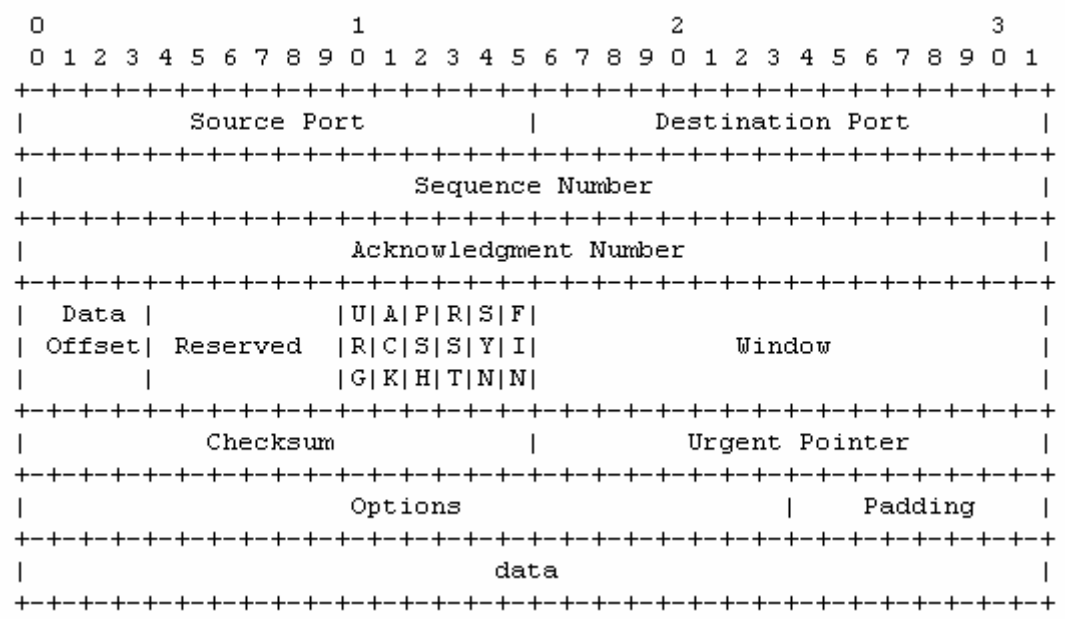

Fig 1.6 Capçalera TCP

Els camps de la capçalera són els següents:

- *Source Port*: Port Origen.
- *Destination Port*: Port Destí.
- *Sequence Number*: Número que identifica al paquet.
- *Acknowledgement Number*: Només s'utilitza quan el flag ACK està actiu. Indica el número de seqüència del pròxim paquet que espera rebre el destí.
- *Data Offset*: Indica la longitud de la capçalera TCP
- *Reserved*: Camp reservat per a un ús futur.
- *Flags*:
  - *URG*: Indica que el punter de dades urgents està actiu
  - *ACK*: Flag del paquet de confirmació. Indica que el camp *Acknowledgement Number* conté dades.
  - *PSH*: Entregar dades immediatament. S'utilitza durant la transmissió
  - *RST*: Resetejar connexió
  - *SYN*: Intent de connexió
  - *FIN*: Finalització de connexió
- *Window*: Mida de la finestra. Indica el total de dades que l'emissor està disposat a acceptar.
- *Checksum*: Control d'errors. Té en compte la capçalera TCP, les dades TCP i una pseudocapçalera que conté les adreces IP.
- *Urgent Pointer*: Aquest camp només s'utilitza quan el flag URG està actiu. Indica quines dades del paquet són urgents.

- *Options* + *Padding*: Conté les dades opcionals del protocol. Si l'opció no ocupa un múltiple de 32 bits, llavors s'han d'afegir bits de farciment (padding) fins a completar la paraula de 32 bits

Al ser un protocol orientat a connexió, es necessiten uns certs passos abans de procedir a enviar les dades. Primer s'ha de realitzar un intent de connexió amb la màquina destí. Aquest s'anomena *Three-Way Handshake* i s'explica més a fons al pròxim apartat. Un cop establerta la connexió, es procedeix a enviar les dades. Durant aquesta transmissió, TCP implementa controls de congestió i de flux per a evitar la saturació dels equips o de la xarxa i mantenir una comunicació el més eficaç possible. Un cop s'acaba la transmissió de dades, es realitza un procés de desconnexió (*Four-Way Handshake*). Aquest procés contempla l'enviament d'un paquet FIN al destí, que aquest confirmarà amb un ACK i enviarà un altre FIN a l'origen, que també respondrà amb un ACK. En aquest moment es dóna per finalitzada la connexió.

1.2.3.1. Three-Way Handshake

En el desenvolupament de la nostra aplicació, totes les proves sobre TCP realitzen un intent de connexió, per tant estudiarem a fons aquest procés.

La connexió comença amb l'enviament d'un paquet SYN al servidor. Si aquesta rep correctament el SYN, llavors envia un SYN/ACK al client, i quan aquest rep el SYN/ACK, tot seguit envia un ACK. Amb aquest tres passos, hem establert una connexió entre les dues màquines.

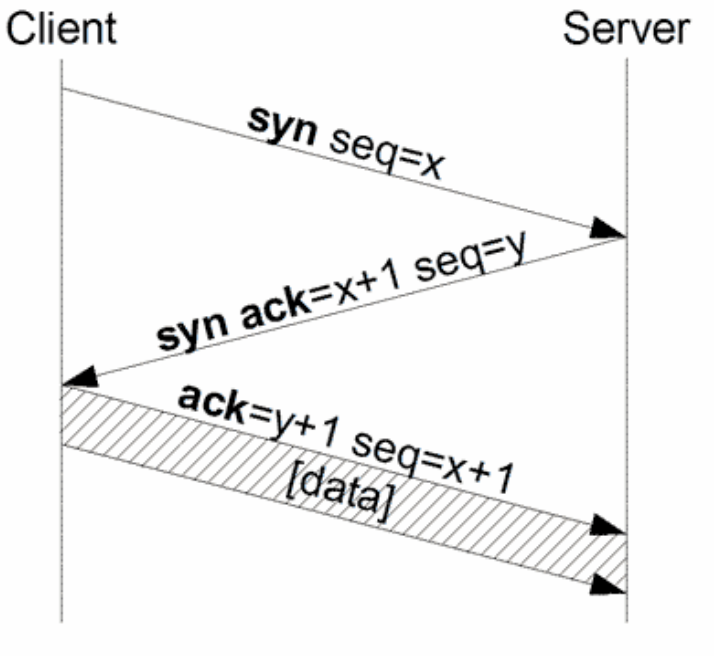

Fig 1.7 Three-Way Handshake

Un detall a destacar del procediment de connexió ve detallat a la figura 1.6. Es tracta dels números de seqüència. Aquests números són diferents per al SYN del client (x) i el del servidor (y), i el ACK, conté el número de seqüència del paquet que confirma més un (SYN(x) → ACK(x+1)), per tant, és imprescindible el seu correcte ús per tal de que la connexió í transmissió de dades es pugui fer sense errors.

CAPÍTOL 2. DETECCIÓ DE LIMITACIONS

2.1. Limitacions en la implementació de piles TCP/IP

Actualment la tendència és que tot dispositiu, per petit que sigui, es pugui connectar a una xarxa, ja sigui per cable o mitjançant una comunicació sense fils. El problema és que, quant més petit és un dispositiu, menys capacitat de recursos té disponibles. Per tant disposa de menys memòria per a guardar dades o una menor velocitat per al processament d'aquestes.

Aquests dispositius tenen certa tendència a utilitzar la pila TCP/IP, al ser la més estesa. Implementar aquesta pila en dispositius tant limitats no és una tasca senzilla, i s'han d'utilitzar tècniques que siguin el més eficaç possible en la optimització de recursos. D'aquestes tècniques d'implementació es destaquen les següents [12]:

- *Zero-Copy*: Es treballa sempre sobre un mateix buffer, de manera que es prescindeix del pas de paràmetres entre diferents entitats del protocol. D'aquesta manera s'estalvia capacitat de processament i memòria.
- *Non-layering*: L'estructuració en capes de la pila afegeix gran quantitat d'*overhead* al procés. S'opta per simplificar l'estructuració i reduir la complexitat del codi.
- *Limitar les funcionalitats dels protocols*: Prescindir de característiques que venen a l'especificació dels protocols, com per exemple, no implementar opcions IP i TCP, o fins i tot, no incloure el mecanisme de control de flux.
- *Limitar les prestacions dels protocols*: Amb l'objectiu de minimitzar l'ús de memòria, s'opta per limitar les connexions TCP simultànies o la longitud màxima dels paquets.
- *Optimitzar la implementació del codi*: Aplicar criteris de disseny dedicats a una arquitectura de hardware concreta, que permetin reduir la quantitat de memòria que ocupen el codi i les dades del programa.

Existeixen ja unes piles adaptades a microcontroladors petits, que permeten establir una comunicació amb aquests dispositius. Una d'elles és la uIP [10], que implementa la pila en 5 KBytes pensada per a microcontroladors de 8 i 16 bits, que a més de IP i TCP, també implementa el ping de ICMP i ARP, però amb nombroses limitacions. Un altre exemple, també pensat per dispositius de 8 i 16 bits, és la lwIP [11], que implementa una pila TCP/IP completa, però simplificada. En aquest cas ocupa 40 KBytes, vuit vegades més que la uIP, però conté els protocols IP, TCP, ICMP , UDP i ARP complets. A la taula 2.1 podem veure una taula de les característiques que implementen cadascuna d'aquestes dues piles.

Taula 2.1. Comparació de uIP i lwIP [13]

Característica	uIP	lwIP
Checksum IP i TCP	X	X
Re-assemblament de paquets fragmentats	X	X
Opcions IP		
Múltiples interfícies		X
UDP		X
Múltiples connexions TCP	X	X
Opcions TCP	X	X
MSS variable (TCP)	X	X
Estimació RTT	X	X
Control de flux (TCP)	X	X
Finestra lliscant (TCP)		X
Control de congestió (TCP)	No necessari	X
Dades Urgents (TCP)	X	X

Com es pot comprovar, les dues piles de les que hem parlat realitzen en certa part el mínim que es demana al RFC 1122 [5], de requeriments d'un host de Internet, encara que hi ha certes característiques que han set obviades per minimitzar l'ús de recursos.

2.2. Detecció de limitacions

A continuació es descriuen els processos a seguir per a determinar unes limitacions bàsiques a tenir en compte, a més de detallar la forma d'actuar dels protocols per poder obtenir-ne les conclusions adients. Part d'aquests tests han estat implementats en l'aplicació que s'ha realitzat, que es descriu al capítol 3 de la present memòria. La finalitat d'aquests tests és intentar determinar en quina mesura està limitada la pila, ja sigui en termes de TCP com de IP, que són els protocols que s'han estudiat més a fons.

2.2.1. Fragmentació IP

L'objectiu de la fragmentació és adaptar la MTU (*Maximum Transfer Unit*) del paquet per a que no superi el longitud màxima de MTU que suporta la xarxa física per la qual ha de viatjar. Això vol dir que la longitud del paquet s'ha d'ajustar automàticament a la suportada per la xarxa.

Fig. 2.1 Escenari on es necessita fragmentació

Com es pot observar a la figura 2.1, tenim un paquet de 1420 bytes que vol anar de l'estació A fins a l'estació C. Això suposa passar per tres xarxes físiques diferents, el problema és que la MTU de xarxa 2 és de 620 bytes. Això vol dir que la trama haurà de fragmentar-se en paquets més petits per poder passar per allí. Aquesta fragmentació la fa el dispositiu que es troba a l'entrada de la xarxa 2. Un cop fragmentat el paquet, aquest continua el seu viatge fins a l'estació C. Cal destacar que el paquet no es torna a ajuntar al passar la xarxa 2, sinó que ja viatja fins al seu destí en forma de fragments i serà el destinatari l'encarregat d'ajuntar-los, reconstruint el datagrama original.

És gràcies a aquesta operació que podrem deduir si té implementada aquesta capacitat. El que haurem de fer en el nostre entorn de proves és generar un paquet i fragmentar-lo en dos o més parts. Llavors, hem d'enviar aquests fragments i comprovar que el dispositiu que estem testejant sap ajuntar correctament els fragments i generar una resposta.

2.2.2. Opcions IP

El protocol IP implementa unes opcions per tal de donar més funcionalitats al protocol. Aquestes opcions es situen al final de la capçalera i cal dir que no són molt utilitzades en l'actualitat, ja que part d'elles també les realitza el protocol ICMP de forma semblant. Tot i així, és un paràmetre que, segons el RFC 1122 [5], tot host ha de saber llegir i interpretar.

El procediment per a realitzar la prova seria enviar un paquet que obligués a la màquina destí a generar una resposta, com un ICMP ECHO REQUEST, i comprovar si aquesta resposta conté els camps d'opcions requerits a la petició anterior. Si és així, es suporten les opcions, si no, el host destí sap llegir el paquet, però ignora el camp d'opcions. Una altra possibilitat és que no es rebi cap resposta, llavors vol dir que la màquina no ha aconseguit llegir el paquet per a generar la contestació.

2.2.3. Checksum TCP

Un dels mitjans més senzills per a determinar la validesa d'un paquet és el control d'errors mitjançant el càlcul del checksum (suma de comprovació). Segons la seva definició, es tracta d'un control de redundància que verifica que

les dades que es reben són correctes (en cas d'error no pot corregir-les). Degut a la seva simplicitat i facilitat d'implementació, és àmpliament utilitzat en xarxes de comunicacions.

En el cas del protocol TCP, intervenen tres parts en el càlcul del checksum: la capçalera TCP, les dades TCP i una pseudocapçalera. D'aquestes parts, la desconeguda fins ara és la pseudocapçalera, que conté part de les dades de la capçalera IP.

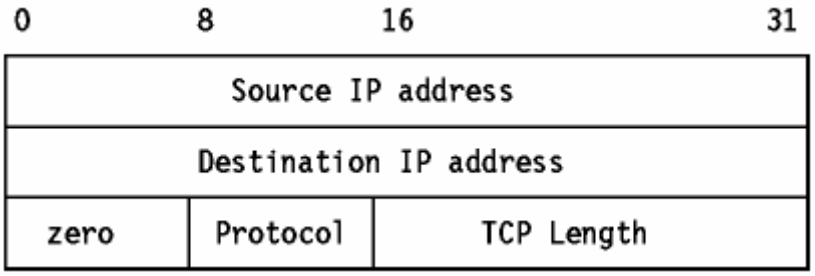

Fig. 2.2 Peudocapçalera per al càlcul del checksum TCP

Com es pot veure a la figura 2.2, la pseudocapçalera està formada per les adreces IP d'origen i destí, el protocol utilitzat (TCP en aquest cas) i la longitud del paquet TCP (capçalera + dades).

Un cop es tenen totes les dades, es realitza el càlcul corresponent, i el resultat s'inclou al corresponent camp de la capçalera TCP.

La prova a realitzar en aquest cas tractaria de comprovar si el dispositiu que estem estudiant realitza el càlcul del checksum per als paquets TCP que rep. Això es pot saber enviant un paquet que contingui intencionadament el checksum incorrecte, llavors caldria esperar una resposta. Si el dispositiu envia una confirmació de rebuda del paquet, llavors vol dir que ha processat les dades i no ha realitzat la comprovació de checksum. En canvi, si no rebem cap resposta, llavors això significa que el paquet ha estat rebutjat i per tant comprova el checksum.

2.2.4. Múltiples connexions TCP

Una connexió TCP s'estableix mitjançant el *three-way handshake*, que s'ha explicat al capítol 1. Amb això el que fem és obrir una connexió per a poder transmetre les dades corresponents. Un cop enviades totes, es finalitza la connexió. El que proposa aquesta prova és descobrir quantes connexions es poden tenir obertes simultàniament.

El procediment a seguir, un cop s'ha trobat un port obert que accepti connexions, és anar establint diàleg amb el dispositiu destí fins que aquest ja no accepti més connexions, és a dir, fins que arribem al punt de que fem una

petició de connexió i aquest ens la rebutgi. En aquest moment, podem dir que la pila ha arribat al seu límit de connexions.

Tot i això, aquest límit no depèn només de la pila, sinó que també es veu afectat per l'aplicació que es troba per sobre de la pila, ja que si aquesta està programada amb un màxim de connexions simultànies, encara que la pila en pugui suportar més, aquestes es rebutjaran. Per tant, la deducció del límit de connexions no va lligada només a la pila, i com a conseqüència, el resultat que es dedueix d'aquesta prova pot no ser totalment fiable per a la detecció de límits de la pila.

2.2.5. Opcions TCP

Les opcions TCP s'inclouen a la part final de la capçalera i com indica el seu nom, al ser opcionals, no és obligatori tenir-les implementades per a que funcioni el protocol TCP/IP correctament. El que fan aquestes opcions és afegir lleugeres millores per a que les transmissions TCP es produeixin de manera més eficaç, o també implementar noves funcionalitats que són útils per al protocol. De totes les opcions que es contemplen actualment al protocol TCP, només 3 van ser definides al primer RFC 793 [3] (End of Option, No Operation i MSS), les altres estan definides al RFC 1323 (*TCP Extensions for High Performance*) [6].

Per al nostre estudi, interessa saber si la pila a provar té implementades les opcions. El cas ideal seria, en un primer pas, provar si es suporten les opcions que es definiren en un principi, i a continuació, comprovar si te implementades les opcions incloses posteriorment.

Segons els RFC consultats, les opcions es defineixen durant el procés d'establiment de connexió, per tant, la idea seria enviar peticions de connexió, cadascuna amb una opció diferent inclosa a la capçalera, i després comprovar si el dispositiu ens ha aconseguit contestar (en aquest cas vol dir que sap llegir el paquet amb opcions) i a més, processar la resposta comprovant si conté les dades que hem demanat al camp d'opcions (llavors vol dir que aquella opció està implementada a la pila).

2.2.6. Estimació RTO

Un dels paràmetres més importants per a obtenir un rendiment òptim en una transmissió TCP és el temporitzador de retransmissió (RTO, *Retransmission Time-Out*). La funcionalitat del RTO és donar un temps màxim d'espera per a rebre la resposta o confirmació d'un paquet que s'ha enviat. Si passat aquest temps, no es rep cap resposta, es procedeix a la retransmissió del paquet. Per a obtenir aquest RTO, es necessita tenir les característiques de la xarxa per la qual es transmet, per tant és necessari conèixer el temps que tarda un paquet des de que surt fins que es rep la resposta, aquest temps és el RTT (*Round*

Trip Time). Durant la transmissió, la pila s'encarrega d'anar estimant el RTT, i si aquest varia, també varia el RTO. Adaptant el RTO a l'estat de la xarxa, es pot optimitzar la transmissió.

Per a obtenir el RTT es poden utilitzar diferents tècniques. La més senzilla seria iniciar un cronòmetre al enviar el paquet i parar-lo al arribar la resposta. Una altra ve definida al RFC 1323 [6], com una nova opció, TCP Timestamp, que permet obtenir exactament el RTT.

El que ens interessa a nosaltres és intentar conèixer el valor dels temporitzadors de la màquina a testejar, especialment del RTO. Per això, necessitaríem provocar una transmissió de dades mitjanament llarga des de la màquina destí fins a la nostra, on tindríem l'aplicació de proves. El que hauria de fer el nostre equip és no enviar algun dels reconeixements de paquets que si hem rebut, amb això provocaríem que expirés el RTO, i per tant obligaríem a fer una retransmissió. El temps que passaria entre el paquet que hem rebut i no hem confirmat, i el que ha retransmès l'equip que estem provant, seria el RTO.

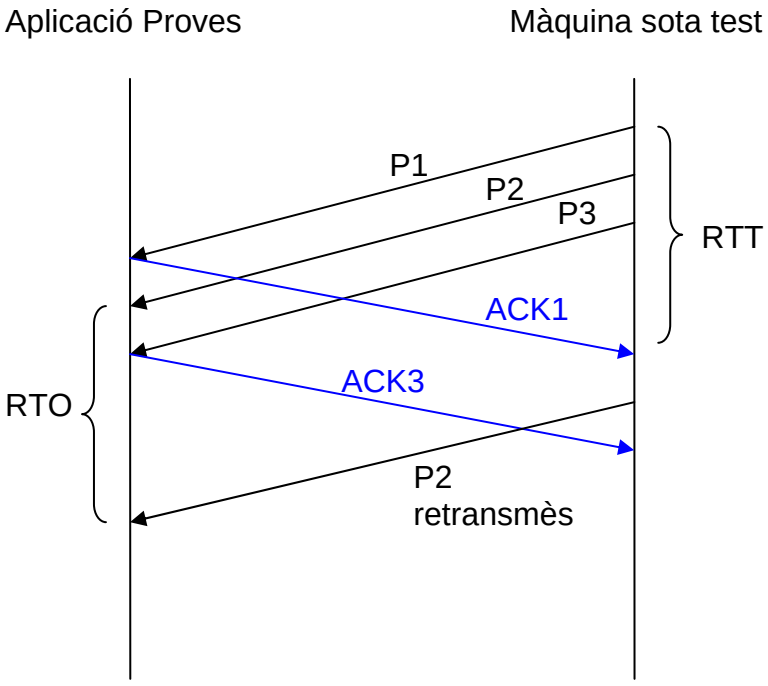

Fig. 2.3 Obtenció RTO

Fent aquesta operació contínuament, també podem saber si la pila que estem provant té els temporitzadors fixes, o si es va adaptant a les pèrdues de paquets i augmenta el RTO.

2.2.7. Control de flux

TCP té un mecanisme per a controlar que un host rebi un nombre de paquets massa gran per a la seva capacitat, el que s'anomena control de flux. D'aquesta manera s'evita saturar al receptor i evitar desbordar els seus *buffers*, amb la conseqüent pèrdua de paquets. Per a això TCP implementa diferents tipus d'algoritmes. El més senzill de tots és el de *stop and wait*, on l'emissor ha d'esperar a rebre la confirmació (ACK) de l'últim paquet enviat per tal de poder enviar el següent. Això implica gran fiabilitat a la transmissió, però poca eficàcia, ja que el flux no és continu, sinó que s'han d'esperar obligatòriament les confirmacions.

L'altre mecanisme de control de flux és el que s'anomena finestra lliscant (*Sliding Window*). L'objectiu d'aquesta finestra, a més d'optimitzar la transmissió, és donar a conèixer a l'emissor el número de bytes que es poden transmetre des de l'últim ACK rebut, d'aquesta manera, aquest pot anar enviant dades sense esperar la confirmació del paquet anterior (dintre del límit que marca la finestra). Es pot dir que *stop and wait* utilitza una finestra que té com a capacitat un paquet.

L'objectiu d'aquesta prova és determinar si l'equip a estudiar si l'equip implementa algun tipus de control de flux. El que s'hauria de fer és obrir una connexió í crear un flux de dades "equip testejat → aplicació de proves". Aquest flux hauria de ser el suficientment llarg com per poder realitzar els tests necessaris per a arribar a les nostres conclusions, per tant necessitaríem, per exemple, la transmissió de dades mitjançant FTP i que ens enviés un arxiu amb prou longitud com per poder fer les proves pertinents.

Un cop establerta la connexió i iniciada la transmissió, mitjançant el camp *Window* de la capçalera TCP, intentarem comprovar la implementació del control de flux. El que farem és anar variant aquest camp, augmentant-lo i disminuint-lo, i comprovar si l'equip adapta el flux a la finestra que nosaltres li indiquem. Si és així, llavors la pila contempla el control de flux, en cas contrari, si ignora les nostres indicacions, no implementa aquesta funció.

2.2.8. Control de congestió

El protocol TCP disposa també de mecanismes per a evitar que una la xarxa per la que han de viatjar els paquets es saturi. Això pot ser degut a que, per exemple, els usuaris de la xarxa tinguin una alta capacitat de transmissió i que els enllaços no disposin de tal capacitat. A causa d'una saturació de la xarxa, la pèrdua de paquets seria gairebé segura. A partir d'aquest problema, es va decidir implementar mecanismes dedicats a controlar la congestió d'una xarxa, amb l'objectiu de mantenir una transmissió fiable en tot moment.

Els mecanismes de control de congestió es basen en realitzar un tanteig sobre la xarxa per intentar esbrinar la capacitat d'aquesta. Ho fan enviant cada cop més cabal de dades i comprovant a partir de quin moment es comencen a

perdre paquets. Dins del control de flux, trobem diferents sistemes, tots definits al RFC 2581 [7], dedicat específicament a la congestió. Aquests mecanismes són: *Slow Start* i *Congestion Avoidance*.

Començant per *Slow Start*, aquest normalment es combina amb *Congestion Avoidance*, degut a que és molt agressiu per a la xarxa. *Slow Start* s'utilitza durant l'establiment de la connexió i el que fa és definir una nova finestra, anomenada finestra de congestió, que defineix la quantitat de dades que poden ser enviades, i la va augmentant fins que es comencen a perdre paquets, llavors el valor de la finestra de congestió es guarda (CWND). Aquest creixement és bastant elevat (exponencial), el que fa que la xarxa es saturi en poc temps, per tant, no convé utilitzar-lo contínuament. La solució és, un cop realitzat el primer tanteig amb *Slow Start,* passar a utilitzar *Congestion Avoidance*, un mètode que realitza una operació similar, però que té un creixement lineal. El procediment és tornar a començar amb *Slow Start* fins que s'arribi a la meitat de la finestra de congestió mesurada en primera instància (CWND/2), i tot seguit, continuar amb *Congestion Avoidance*. A la figura 2.4 es pot veure aquest procés clarament.

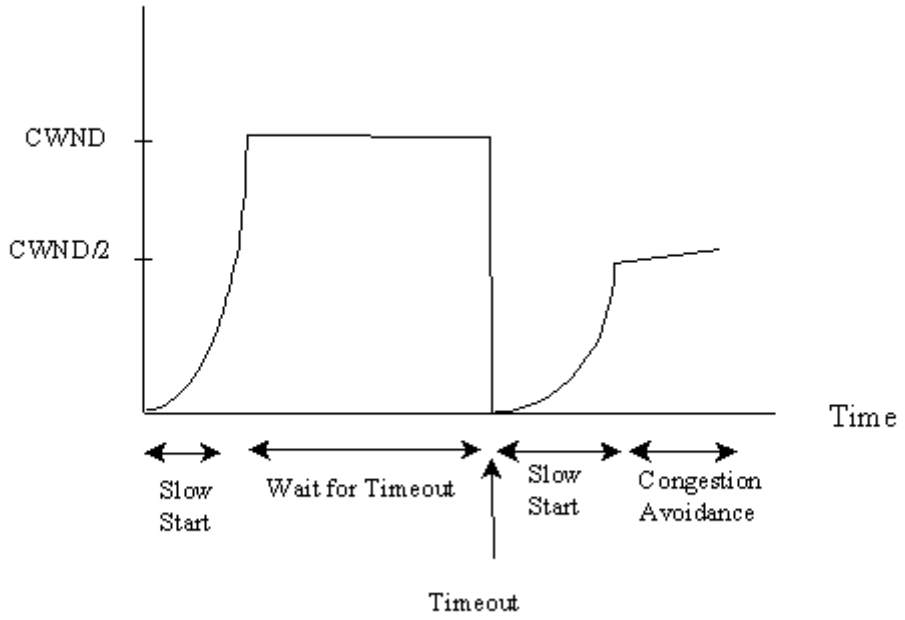

Fig. 2.4 Slow Start i Congestion Avoidance

En la nostra prova es vol comprovar si la màquina limitada te implementades a la seva pila aquests mecanismes. Per a poder realitzar el test, hem de crear un flux "equip testejat → aplicació de proves", igual que a la prova anterior.

Un cop tinguem el flux establert, es procedeix a observar com evoluciona el tràfic per a deduir el control de congestió que implementa, si és que n'implementa algun. El que hem d'estudiar és com varia el flux de dades, més concretament la longitud de la ràfega de paquets, si aquesta és la mateixa durant tota la transmissió o si augmenta o disminueix degut a l'acció dels algoritmes.

Tot i això, observar aquesta evolució no és del tot evident, ja hauríem de controlar més paràmetres, a més de comptar el número de paquets que ve en cada ràfega. El problema és que si quan arriba un paquet, de seguida s'envia la confirmació ACK, serà impossible diferenciar les ràfegues, ja que aquestes formarien un flux continu i no es podria saber quins paquets formen part de cadascuna. Suposant que tenim un enllaç directe "origen→destí", on tenim un retard mitjanament estable, el que hauríem de fer és retardar l'enviament de l'ACK del primer paquet de la ràfega fins que ens no ens arribin tots els paquets. En aquest moment, obtenim el número de paquets de la ràfega i enviem l'ACK. Per fer això, s'ha de tenir molt en compte el RTO del destí, ja que si aquest expira, llavors es provocaria la retransmissió i no es podria portar a terme la mesura.

Aquesta prova, a més de crear el flux "monitor→equip testejat", té el problema afegit de gestionar els temporitzadors, cosa que la fa pràcticament poc viable.

CAPÍTOL 3. DESENVOLUPAMENT DE L'APLICACIÓ

Aquest tercer i últim capítol està completament dedicat a l'aplicació de detecció de limitacions de piles TCP/IP que s'ha implementat. En un primer terme es descriuen les característiques del disseny, els objectius que es volien assolir amb aquest programa i tot el procés que s'ha seguit per al seu desenvolupament. A continuació es passa a descriure cadascuna de les proves que s'han implementat a l'aplicació, els passos que segueixen per a obtenir la informació i les conclusions que s'extreuen d'aquestes dades.

3.1. Característiques de disseny

Amb aquest apartat entrem ja a la part de l'aplicació que s'ha implementat. Aquí es descriuran els conceptes generals que formen part del desenvolupament de l'aplicació.

En un principi, la idea del disseny anava enfocada a crear una aplicació sobre Windows, que mitjançant un conjunt de proves, pogués donar certes dades que ens permetin tenir una idea de les capacitats d'una pila TCP/IP d'un equip remot. Per tant, seguint aquest marc d'aplicació, es va dissenyar un software amb un motor capaç d'enviar i rebre paquets, i a més, que sap obtenir unes conclusions processant les respostes rebudes.

L'aplicació utilitza el llenguatge C++, més modern que el C clàssic i amb més funcions dedicades a les comunicacions en xarxa. Per al motor d'enviament i captura de paquets, es va optar per la utilització de les llibreries winPcap, conegudes per què són les que utilitza el capturador de paquets Ethereal. Varem decidir optar per aquestes llibreries després de realitzar proves amb els WINSOCKS, l'apartat de comunicacions de C++, que no resultaren satisfactòries, ja que no complien els nostres requisits.

En quant a la interfície gràfica, es va valorar la utilització la MFC, pròpia del compilador de Microsoft, o fer ús de la llibreria *open-source* wxWidgets. Al final ens varem decantar per l'ús de wxWidgets, ja que era lleugerament més senzilla d'implementar i a més oferia un suport multi-plataforma.

A més de totes aquestes llibreries afegides, també es va requerir actualitzar les llibreries de C++ a la seva última versió, per al correcte funcionament de WinPcap i wxWidgets (WinpCap requeria tenir la segona versió dels WINSOCK). Tot aquest conjunt d'actualitzacions i llibreries funcionant simultàniament va provocar molts problemes de redefinició de variables a varies d'aquestes llibreries. La forma de solucionar-ho va ser definint-les només on era estrictament necessari i seguint aquest ordre: WINSOCK2 → WinPcap → wxWidgets.

3.1.1. Llibreries WINPCAP

WinPcap és una conjunt de llibreries que permeten obtenir accés a una xarxa a nivell d'enllaç sobre sistemes Windows. Permet crear aplicacions per a capturar i transmetre paquets sobre una xarxa evitant la pila de protocols que té implementada el sistema operatiu.

A més de les llibreries, també inclou un driver que és el que proporciona accés a la xarxa en els seus nivells més baixos. Aquest driver permet tenir comunicació directa amb la interfície de xarxa.

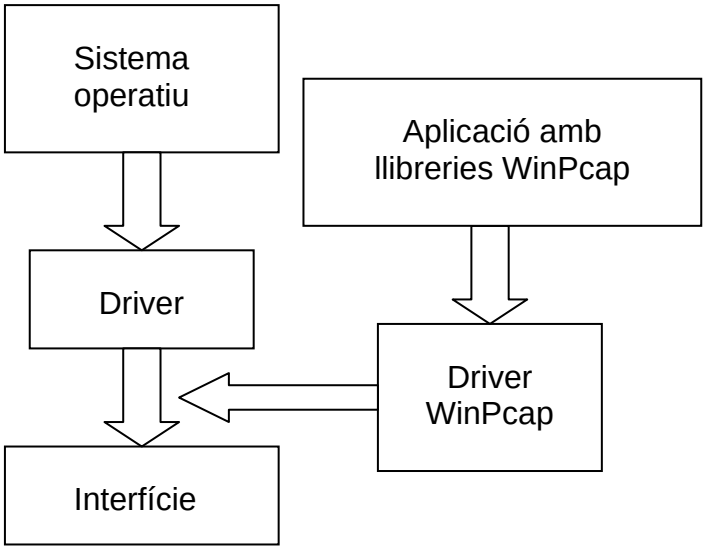

Fig. 3.1 Esquema de la organització de WinPcap

Això no vol dir que WinPcap pugui interferir en el tràfic que passa per la interfície, simplement es dedica a "escoltar", i mitjançant el seu propi driver, donar accés a l'aplicació a aquestes dades. Per tant, WinPcap no podria ser utilitzat per a programar aplicacions que tinguin que bloquejar tràfic, com un firewall per exemple.

Per a la nostra aplicació, la utilització de WinPcap, al cap i a la fi, és millor que fer-ho mitjançant els WINSOCK, ja que podem tenir accés a la xarxa en un nivell més baix (hem d'especificar les adreces físiques, cosa que no s'ha de fer amb els sockets de Windows) i podem capturar tots el paquets que arribin sense por de que el sistema operatiu ens bloquegi aquest tràfic. Com a últim detall, dir que hem utilitzat la versió 3.1 de WinPcap, la darrera versió estable que existeix, encara que s'està treballant amb una nova versió, la 4.0, actualment en fase de proves.

3.1.1.1. Funcions WinPcap

Les llibreries WinPcap contenen gran quantitat de funcions, bona part d'elles dedicades a la captura de paquets. Aquí ens centrarem en les que hem utilitzat en la nostra aplicació.

```
pcap_findalldevs(&alldevs, errbuf)
```

Aquesta primera funció, `pcap_findalldevs`, es dedica a buscar les interfícies de xarxa que té la nostra màquina, per tal d'escollir-ne una, que serà la que utilitzarem per a capturar o enviar els paquets. A dins de l'estructura `alldevs` es guarden les dades de totes aquestes interfícies. Cada interfície conté paràmetres, com el nom o la descripció i venen definides a dins d'una estructura que hem anomenat `d`. Per tant, `alldevs` és una estructura, que a la vegada conté una estructura, `d`, per a cada interfície que s'ha trobat.

```
fp = pcap_open(d->name,
               200,
               PCAP_OPENFLAG_PROMISCUOUS,
               1000,
               NULL,
               Errbuf
               )
```

Un cop tenim totes les interfícies buscades, hem d'escollir la que volem utilitzar i obrir-la. Això es fa amb la funció `pcap_open`, a la que li indiquem el nom de la interfície, el número de bytes que es volen capturar de cada paquet, el mode d'utilització de la interfície i un timeout de lectura. Aquesta interfície es quedarà guardada a l'estructura `fp`, que serà la que utilitzarem a partir d'ara.

```
pcap_sendpacket(fp,
               paquete,
               sizeof (struct iphdr) + sizeof (struct
               tcphdr)+14
               )
```

La funció que tractem a continuació és la dedicada a enviar un paquet. Com a paràmetres, hem de passar la interfície, el buffer del paquet a enviar i la longitud d'aquest buffer.

```
pcap_next_ex( fp, &header, &pkt_data)
```

Per a rebre paquets, s'utilitza la funció `pcap_next_ex`, a la que li donem com a paràmetres la interfície, unes capçaleres que crea la llibreria (no les capçaleres IP i TCP), i un buffer on guardarà el paquet que s'ha rebut, amb capçaleres i dades.

3.1.2. Llibreries wxWidgets

Per a la realització de la interfície gràfica (GUI, Graphic User Interface), varem decidir fer us d'un conjunt de llibreries *Open-Source* anomenat wxWidgets (anteriorment wxWindows). La principal característica és que aquesta GUI és multi-plataforma, és a dir, que funciona tant en Windows com a Unix o MAC. Això permet la creació de programes sense la limitació del sistema operatiu en el qual seran utilitzats. A més, proporciona una API (Application Programming Inteface, Interfície de Programació d'Aplicacions) senzilla e intuïtiva que facilita la creació de codi.

3.1.2.1. Funcions wxWidgets

A les llibreries d'aquesta interfície gràfica es defineixen tot tipus de classes per a diferents objectes que es poden trobar a una finestra, com un botó, un menú o un quadre de text. El que explicarem en aquest apartat va més dedicat a com s'han d'interpretar les accions que es realitzen sobre aquests objectes, com per exemple, pitjar un botó.

El primer que s'ha de fer és definir el botó:

```
BTN_Aceptar = new wxButton(PNL_fondo, BTN_OK, "Aceptar",
                           wxPoint(200,144), wxSize(100,22));
```

A la definició, es donen paràmetres com el text que mostrarà el botó, la mida d'aquest botó o la situació a la finestra. En el nostre cas, ens interessa fixar-nos en el identificador `BTN_OK`.
Tot seguit, el que s'ha de fer és declarar aquest l'acció d'aquest botó a la taula d'accions (*Event Table*). Aquesta taula conté totes els accions que es poden realitzar a la finestra.

```
BEGIN_EVENT_TABLE(Projecte, wxFrame)

     EVT_BUTTON(BTN_OK, Projecte::OnBotonAceptar)

END_EVENT_TABLE()
```

Veiem que a la definició de l'acció, es dóna l'identificador de l'objecte i la funció que executa aquest botó.

```
void Projecte::OnBotonAceptar(wxCommandEvent &event)
{
     //codi de la funció
}
```

Un cop tenim definida la funció, només s'ha d'incloure a dins el codi que s'ha d'executar al pitjar aquest botó.

3.2. Descripció de l'aplicació

Aquest segon apartat va dedicat a descriure el funcionament de l'aplicació, donant les pautes per al seu ús correcte. També s'expliquen quines de les proves del capítol 2 s'han implementat i es fa un estudi del cadascun dels passos que segueixen des que es crida a la funció fins que s'obté la conclusió final.

En aquesta descripció no s'entrarà en explicar específicament el codi de l'aplicació desenvolupada. El que es pretén és donar a conèixer els procediments que s'han seguit per a aconseguir passar la base teòrica de les proves del capítol 2 a una aplicació de software que ofereixi aquestes operacions. Per això, es descriuen els blocs més importats en els que es divideix l'aplicació, i s'expliquen quines són les funcions més importats de cadascun d'ells.

3.2.1. Creació de paquets

Una part important de l'aplicació és poder crear els paquets necessaris per a fer les proves. El primer pas és definir unes estructures que continguin les dades de les capçaleres. Per a crear aquestes estructures, el més importat és tenir en compte la mida en bytes de cada camp de la capçalera del protocol, per així poder definir quines variables en formen part.

```
struct tcphdr {
 unsigned short int th_sport;
 unsigned short int th_dport;
 unsigned int th_seq;
 unsigned int th_ack;
 unsigned char th_x2:4, th_off:4;
 unsigned char th_flags;
 unsigned short int th_win;
 unsigned short int th_sum;
 unsigned short int th_urp;
};
```

L'estructura que es mostra és la de la capçalera TCP. Si es compara amb la figura 1.6, es pot veure que cada camp de la capçalera ve definit per tipus de variable, depenent de la seva longitud.

Un cop es tenen les estructures definides, cal incloure-les a dins d'un buffer, d'aquesta manera es forma un paquet.

```
struct iphdr *iph;
iph = (struct iphdr *) (paquete + 14);

struct tcphdr *tcph;
tcph = (struct tcphdr *) (paquete + sizeof (struct iphdr) + 14);
```

El que fan les sentencies anteriors és apuntar a les estructures de dades de la capçalera IP (*iph*) i TCP (*tcph*), i les inclou a dins d'un buffer anomenat *paquete*. D'aquesta manera, per omplir el paquet només s'ha de passar la dada corresponent i guardar-la al camp de l'estructura seleccionada. Per exemple, si volem que el port de destí sigui el 80, llavors hem de fer el següent:

```
tcph->th_dport = htons(80);
```

A l'hora de omplir les dades, s'ha de tenir en compte la forma com es guarden aquestes dades. Hi ha dades que, depenent de la màquina, es guarden amb el bit més significatiu a un costat o a un altre. Això pot causar que a l'hora d'enviar el paquet, els bits es trobin al revés i la màquina destí no els entengui. Per això, els protocols tenen una manera definida d'enviar les dades, i per passar les nostres dades al format de dades de xarxa, s'utilitza la funció `htons()`.

Cada prova s'encarrega de omplir les estructures amb les dades del paquet a enviar i a més inclou aquestes estructures a un buffer que farà les funcions de paquet.

3.2.2. Enviament i rebuda de paquets

Per a enviar i rebre paquets, s'utilitzen les funcions `pcap_sendpacket` i `pcap_next_ex` que ja s'han explicat a l'apartat 3.1.1. El que no s'ha explicat és la forma de processar els paquets un cop s'han rebut. En aquest cas es realitza el procediment invers a la creació de paquets, ja que partim d'un paquet i el que volem és que les dades d'aquest paquet es guardin a les nostres estructures. Per aconseguir això, el procés a seguir és el mateix que s'ha definit a l'apartat anterior

```
struct iphdr *ip_rcv;
ip_rcv = (struct iphdr *) (rcv_pkt + 14);

struct tcphdr *tcp_rcv;
tcp_rcv =(struct tcphdr *)(rcv_pkt + sizeof(struct iphdr) + 14);
```

D'aquesta manera, apuntem les nostres estructures a la paquet rebut, i les capçaleres agafen les dades del paquet. La única diferència és que per a entendre les dades haurem d'utilitzar la funció `ntohs()`, l'inversa de `htons()`.

```
port = ntohs(tcp_rcv->th_dport);
```

3.2.3. Deteccions implementades

Quan es va dissenyar l'aplicació, es va pensar en les proves que es podrien fer sense que nosaltres tinguéssim accés a la màquina a provar, és a dir, que no podem donar-li ordres a la màquina destí. Per tant, de les proves explicades al

capítol 2, les que necessitin crear un flux "màquina destí→aplicació de proves", com el cas del test d'estimació de RTT, no s'implementaran.

Per a cada prova, es detallaran les operacions que es realitzen pas a pas i es mostraran diagrames per tal de que les explicacions siguin més comprensibles.

3.2.3.1. Test de protocols

Aquest primer test està pensat per a realitzar-lo cada cop que es vol provar un equip nou. El que fa és donar a conèixer quins protocols té implementats la màquina destí. Les proves es realitzen sobre els protocols ICMP i TCP, que són els que utilitza el software de proves.

A més, també fa una prova per saber si UDP es troba implementat, però, al ser UDP un protocol no orientat a connexió i que no assegura l'enviament de les dades, aquest no te cap mecanisme de confirmació, per tant, per poder realitzar el test de UDP seria precís que la maquina a estudiar tingués implementat alguna eina com un servidor UDP Echo que ens confirmés l'arribada del paquet.

Entrant més a fons en el funcionament de la prova, per a cada protocol s'utilitza un tipus de diàleg diferent. En el cas de ICMP, es fa ús del que es coneix com ping, s'envia un paquet ICMP ECHO REQUEST i s'espera l'arribada d'un ICMP ECHO REPLY. Si tot és correcte, vol dir que el protocol ICMP està implementat.

En quant al TCP, la prova es realitza mitjançant un intent de connexió. Des del software de proves s'envia un paquet SYN al destí, i s'espera resposta. En aquest cas podem rebre dues contestacions diferents, un SYN-ACK si és possible establir connexió o un RST-ACK si ens rebutja la connexió (segurament perquè el port es troba tancat). En qualsevol dels casos, al rebre resposta vol dir que la pila de l'equip destí té el protocol TCP implementat, si no es rep resposta, doncs llavors significa que no té funcionalitats TCP.

Com a últim detall, destacar que si no s'ha realitzat aquest test al començament, al intentar realitzar una altra prova, el programa ens avisarà dient que es desconeixen els protocols que suporta el destí, i automàticament realitzarà un test del protocol que necessiti la prova escollida (TCP o ICMP). Si els resultats són satisfactoris, es podrà procedir a fer la prova, si no, no podrem ja que l'equip no suporta aquell protocol.

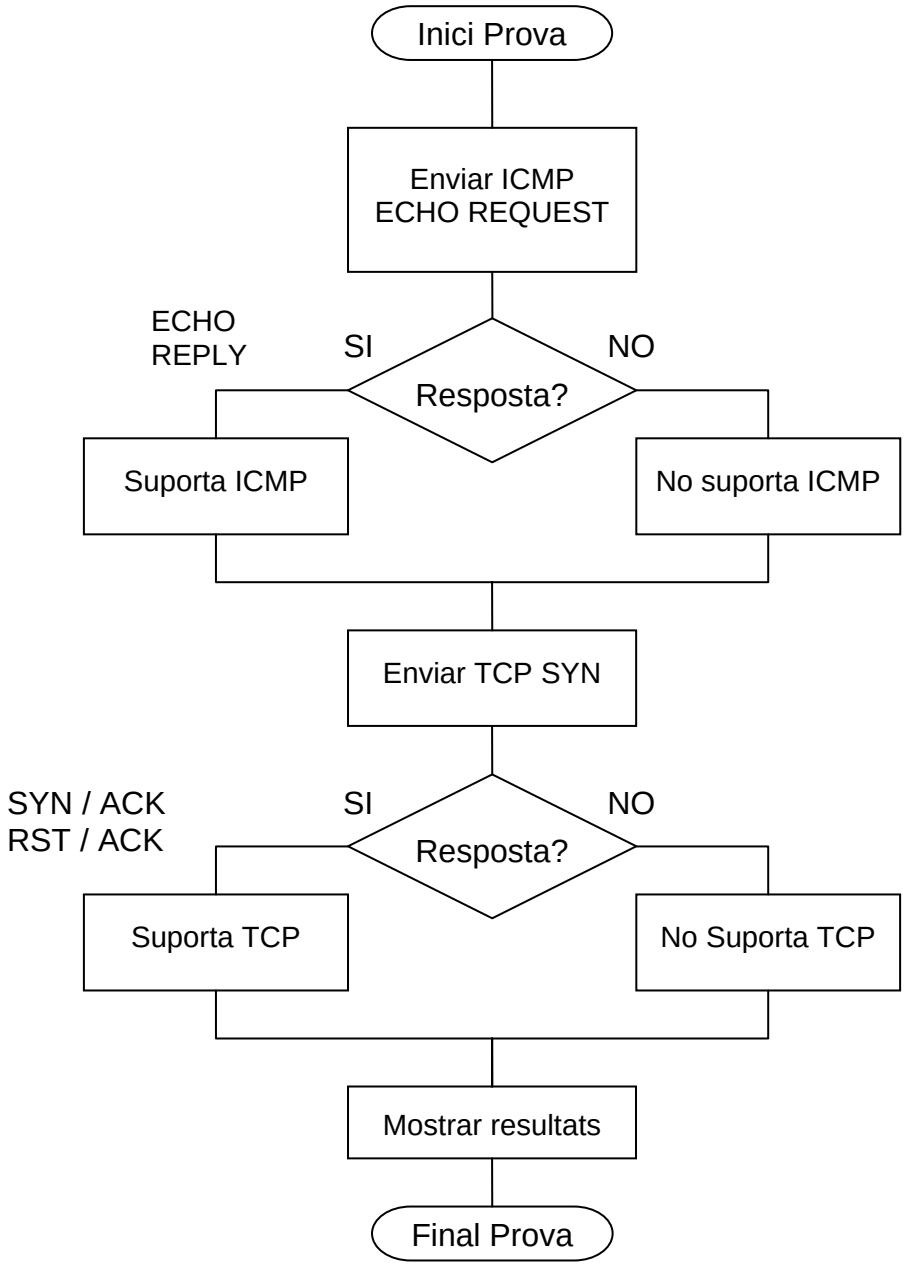

Fig. 3.2 Diagrama de flux de la prova Test de Protocols

3.2.3.2. Scan de ports

Hi ha una sèrie de proves que precisen realitzar una connexió TCP completa, i per tant, que necessiten com a mínim un port obert a la màquina destí. A causa d'això, s'ha optat per a incloure una funció que realitza un escaneig de ports sobre l'equip destí. La prova es realitza als port coneguts (*well-known ports*) que la IANA té definits des del 0 fins al 1023.

El procediment per a descobrir els ports oberts es basa en la connexió TCP. S'envia un paquet SYN, i si la resposta és SYN/ACK, el port està obert, en canvi si la resposta és RST/ACK o no s'obté resposta, el port es troba tancat. Aquest procediment és el que ve descrit a la figura 3.3.

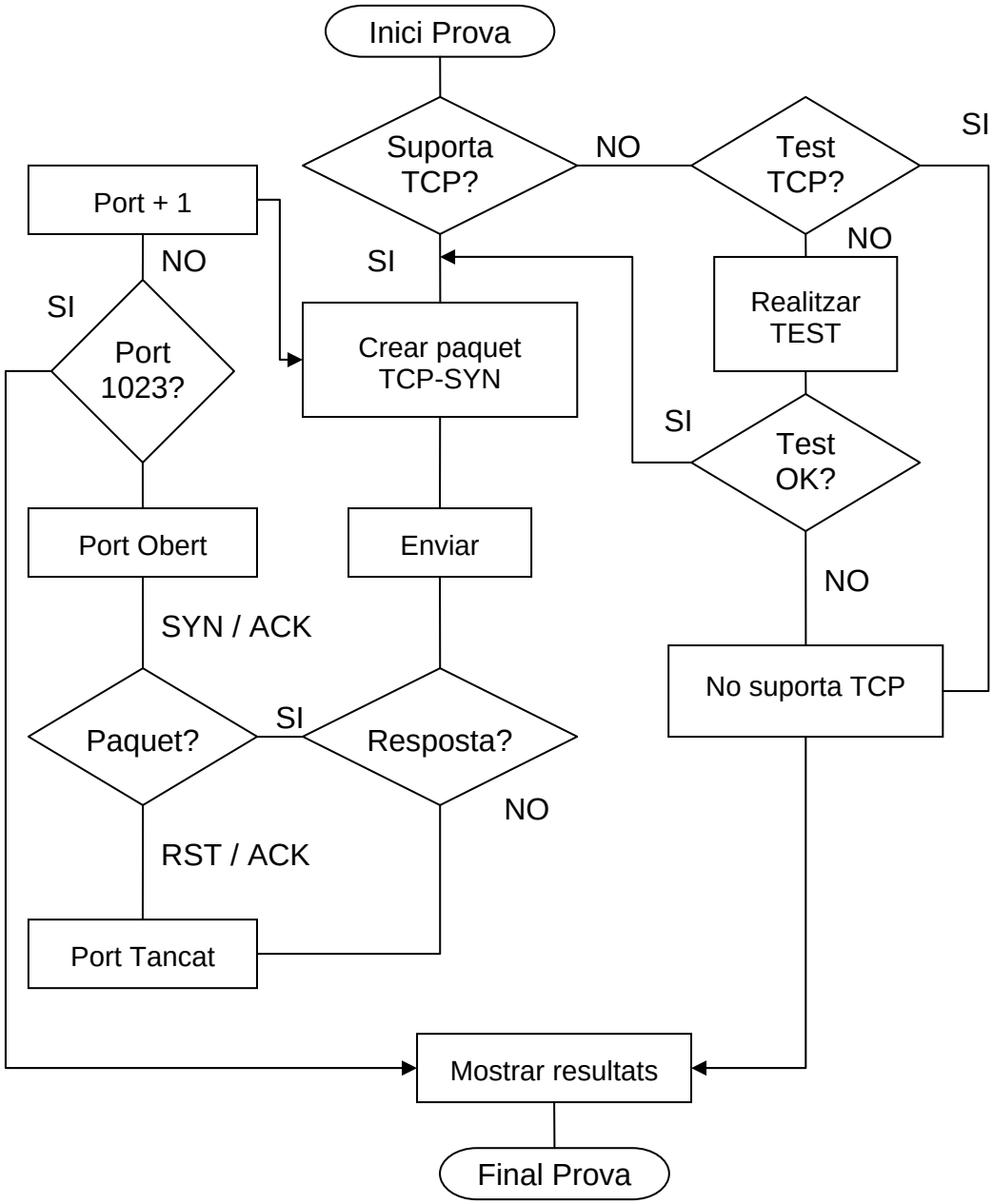

Fig. 3.3 Diagrama de flux de la prova Scan de ports

3.2.3.3. Fragmentació IP

Aquesta segona prova està pensada per a saber si la maquina a estudi implementa les funcions de fragmentació i re-assemblament de paquets del protocol IP. La prova segueix les pautes de l'explicat al capítol 2, per tant entrarem directament a descriure com s'ha implementat a la nostra aplicació.

En aquest cas s'ha utilitzat el protocol TCP per a fer les proves. El procediment que es segueix comença amb la creació d'un paquet de connexió TCP, TCP-SYN, que en comptes d'anar directament encapsulat dins d'un paquet IP, el que farem serà dividir-lo en dues parts i enviarem dos paquets IP, cadascun amb un fragment del SYN.

Una cosa a destacar és que cada fragment que creem porta una capçalera IP completa, però alguns dels seus camps contenen dades sobre la fragmentació. Aquests camps són:

- Fragment 1:
  - Flag: MORE FRAGMENTS
  - Fragment Offset: 0

- Fragment 2:
  - Flag: 0
  - Fragment Offset: 8

Es pot veure com varien els camps depenent de quin fragment estudiem. Al primer veiem que el flag MORE FRAGMENTS està activat i l'offset és 0, per tant això vol dir que ens trobem al primer fragment. El segon ens indica un offset de 8 i cap flag actiu. Això ens indica que és l'últim fragment i que a l'hora de muntar el paquet, les dades d'aquest segon fragment es troben a partir del byte 8 del paquet total.

Com a últim apunt, comentar que en un principi es pensava fer aquesta prova amb el protocol ICMP, mitjançant el ECHO REQUEST – ECHO REPLY, però no va poder ser degut a que el mínim paquet que es pot fragmentar és de 8 bytes, exactament el que ocupa la capçalera ICMP, per tant es va fer amb TCP, que té 20 bytes de capçalera.

El procediment seguit es veu a diagrama de la figura 3.4, on observem que el primer de tot és comprovar si es suporta el protocol sobre el que volem testejar, un cop feta la prova prèvia, es passa al test de fragmentació. Si s'obté una resposta, llavors la màquina destí sap ajuntar paquets fragmentats correctament per tant suporta fragmentació. En el cas de no rebre cap tipus de resposta, es dedueix que no ha aconseguit llegir el paquet i per tant que no suporta fragmentació.

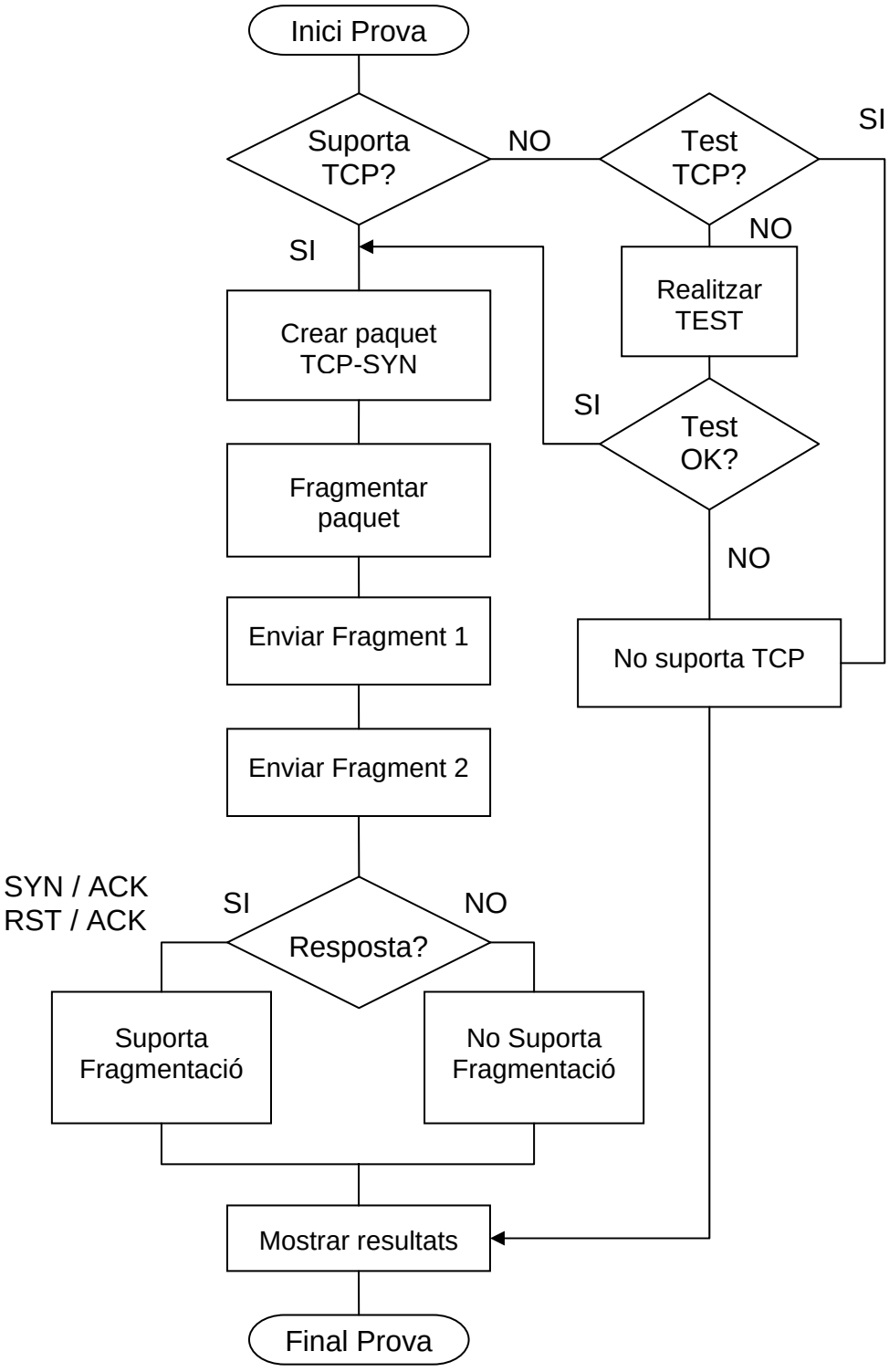

Fig. 3.4 Diagrama de flux de la prova Fragmentació IP

3.2.3.4. Opcions IP

Aquesta prova es dedica a comprovar si la màquina destí te implementades les opcions IP a la seva pila. Com ja s'ha comentat a l'apartat 2.2.3, aquest tipus d'opcions no està molt estès, però segons el RFC 1122 [5] de requeriments d'un host de Internet, tota màquina ha de saber llegir-les com a mínim. Aquí, a més de saber si la màquina sap llegir bé els paquets també es vol provar si aconsegueix respondre i omplir correctament.

Per al nostre test, s'utilitzarà la opció IP TIMESTAMP. Aquesta opció està pensada per a mesurar el retard de transmissió que existeix entre dues màquines. Per això, el que es fa és enviar a dins de la capçalera IP, una marca de temps que indica l'hora exacta a la qual ha sortit el paquet. Llavors, la màquina que rep aquest paquet compara l'hora de sortida amb l'hora d'arribada i s'obté el retard corresponent. El format amb que viatja aquest temps és en mil·lisegons, concretament, el que s'envia és el total de mil·lisegons que han transcorregut des de la mitjanit GMT. El problema d'aquesta opció és que, per a donar uns resultats satisfactoris, tots els elements de la xarxa haurien d'estar sincronitzats al mateix rellotge, cosa quasi impossible.

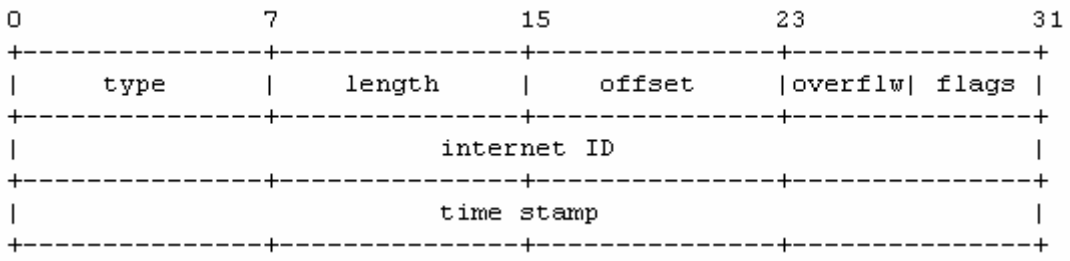

Fig. 3.5 Capçalera de la opció IP Timestamp

A la figura 3.5 es poden veure els camps que formen la capçalera de l'opció. Els que més ens interessen són el *internet ID*, que emmagatzema la IP, i el *time stamp*, que guarda el temps en mil·lisegons.

En aquest cas, utilitzarem el protocol ICMP, fent us del ping. Com a totes les proves, es comença per comprovar si el destí suporta el protocol i després s'envia un paquet ICMP ECHO REQUEST amb la opció Timestamp a la capçalera IP, llavors s'espera la resposta. Si la contestació ECHO REPLY arriba, l'equip destí ha aconseguit llegir el paquet amb opcions i, si a més aquesta resposta també conté la opció amb el timestamp, vol dir que té implementada la opció. A la figura 3.6 es pot veure el diagrama de flux d'aquesta prova.

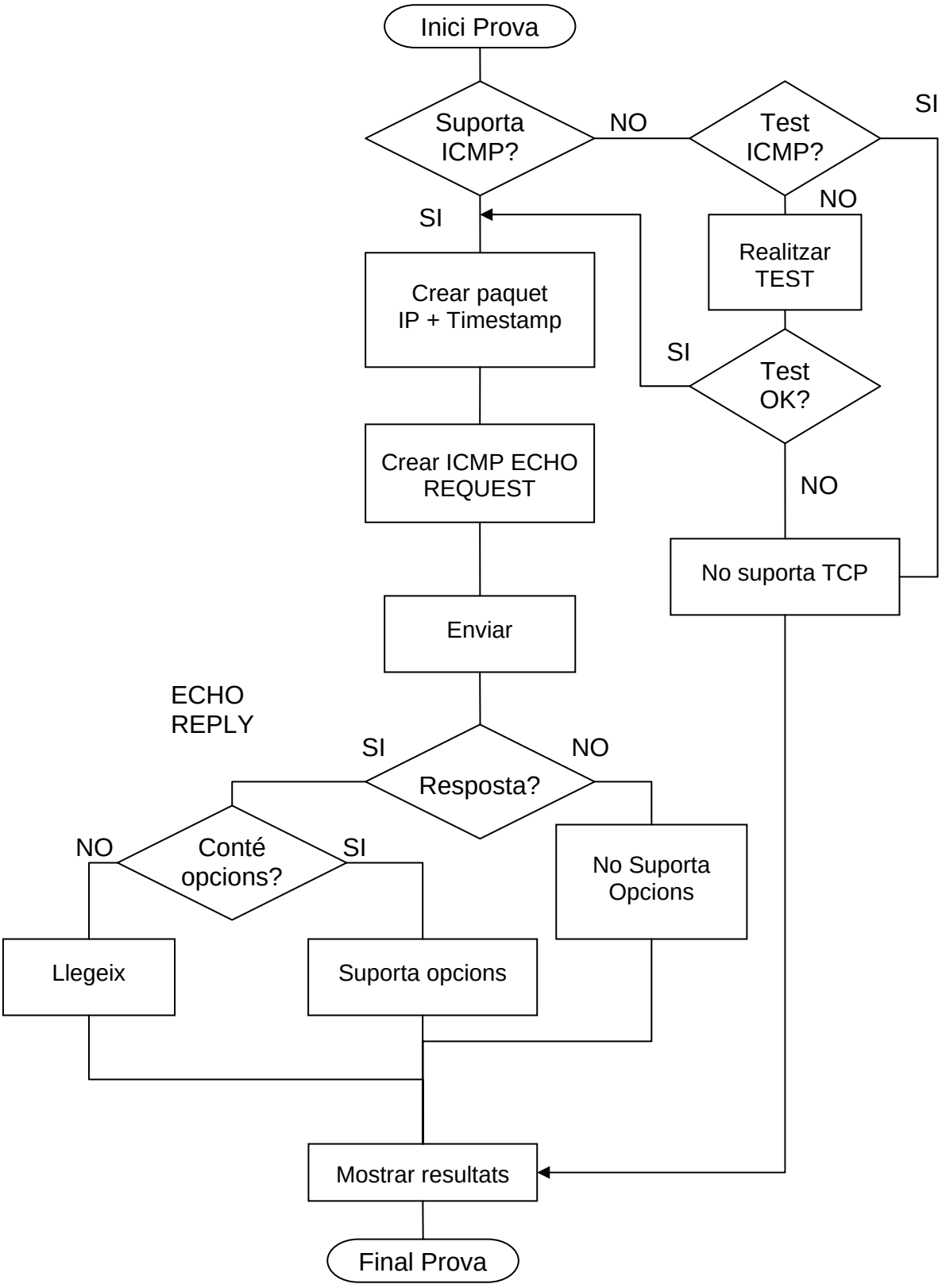

Fig. 3.6 Diagrama de flux de la prova Opcions IP

3.2.3.5. Checksum TCP

Amb aquesta prova es vol saber si l'equip destí realitza el càlcul de checksum TCP per a comprovar si el paquet és correcte o no. La teoria d'aquest càlcul ja s'ha explicat al capítol 2, per tant passarem directament a explicar com s'ha implementat a la nostra aplicació.

Per començar, en la realització d'aquesta prova s'envia un paquet TCP-SYN d'establiment de connexió. El que es fa és realitzar dos intents de connexió, el primer és un paquet SYN normal, simplement de comprovació, per a assegurar que funciona el protocol TCP. El segon és el paquet de prova, que s'envia amb un checksum incorrecte. L'objectiu d'aquest paquet és comprovar si la màquina destí llegeix i processa aquest paquet, o pel contrari, el rebutja.

Per tant si el segon paquet rep una contestació SYN-ACK o RST-ACK, llavors es pot deduir que la pila ignora el checksum i no el comprova. Si no es rep aquesta resposta, llavors sabem que el paquet ha set rebutjat a causa de que té implementada la comprovació de checksum. A la figura 3.7 es mostra el diagrama de flux d'aquesta prova.

3.2.3.6. Connexions TCP

L'objectiu d'aquest test és intentar descobrir quantes connexions simultànies pot mantenir obertes la pila de l'equip destí. Per a realitzar aquesta prova també farem ús del *three-way handshake* d'establiment de connexió TCP.

En al nostra implementació, representada a la figura 3.8, el que fem és demanar a l'usuari el número d'intents de connexió que vol realitzar, i a partir d'aquí, anirem enviant paquets SYN fins que la màquina destí ens rebutgi la connexió (RST-ACK). Un detall molt important és que en aquest cas, és obligatori disposar d'un port obert a l'equip que estem estudiant per poder establir la connexió, si no és així, la prova no es pot realitzar, ja que si ens rebutja els intents de connexió des d'un principi, no podem estimar el límit de la pila. Un altre detall, és que per a cada paquet que s'envia, hem de canviar el port de sortida i el número de seqüència, degut a que no podem establir dues connexions si sortim del mateix port. El que fem és començar pel port que ens ha indicat l'usuari des d'un principi, i per a cada nou SYN, augmentem el número de port en 1.

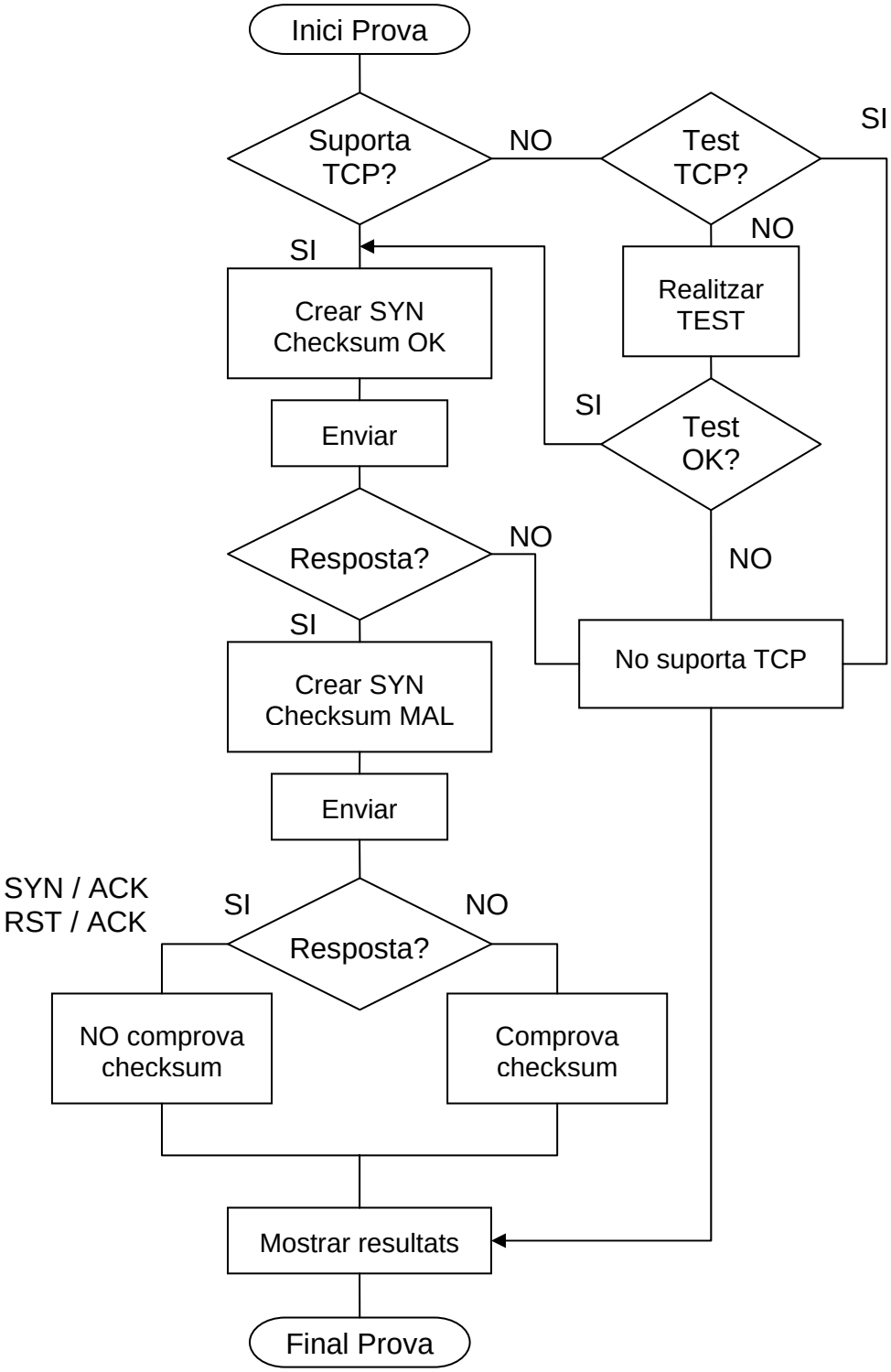

Fig. 3.7 Diagrama de flux de la prova Checksum TCP

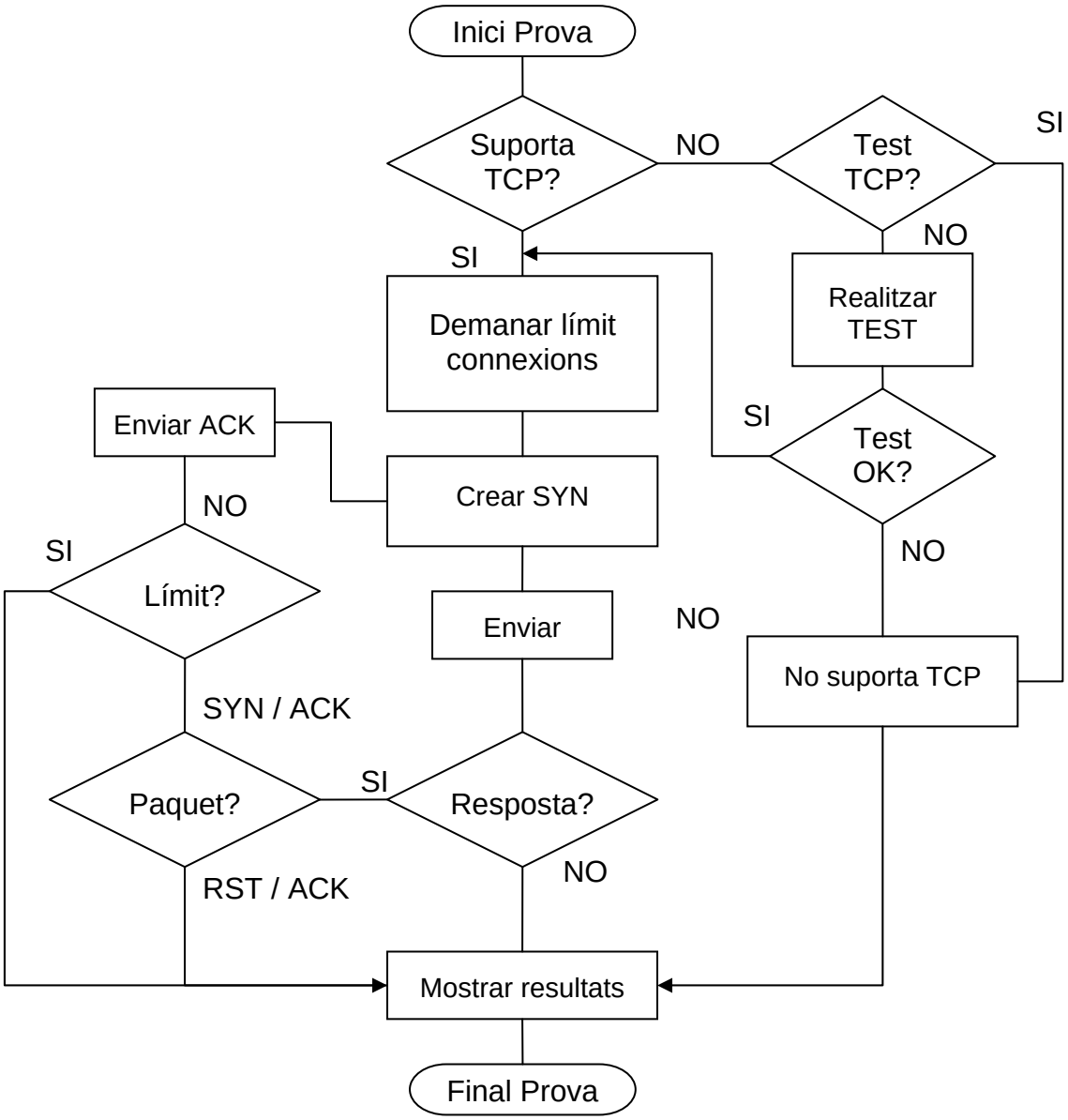

Fig. 3.8 Diagrama de flux de la prova Connexions TCP

3.2.3.7. Opcions TCP

En el cas de la prova d'opcions TCP es realitzarà un procés semblant al de les opcions IP, però en aquest cas, en comptes de ICMP, estem obligats a utilitzar TCP.

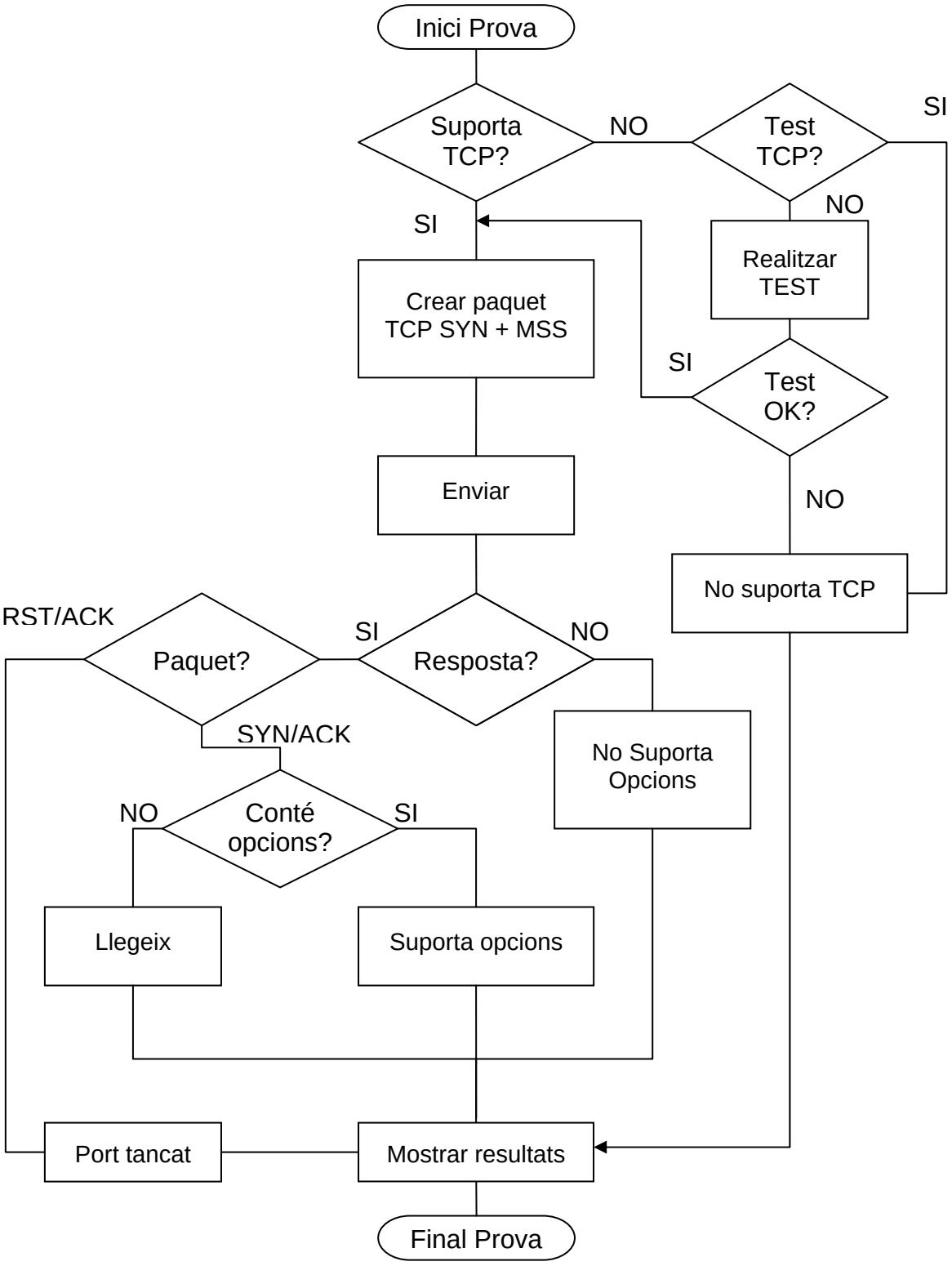

Fig. 3.9 Diagrama de flux de la prova Opcions TCP

Per al nostre test, farem ús d'una de les opcions que més utilitzades, el MSS (*Maximum Segment Size*). Aquesta opció és una de les que venia definides al primer RFC de TCP, i el seu objectiu és definir la longitud màxima que pot tenir

el camp de dades de TCP. El protocol TCP utilitza aquesta opció per a ajustar la longitud de MSS per a que el paquet pugui ser enviat sense fragmentar, és a dir, que adapta la longitud de MSS a la MTU de la xarxa, d'aquesta forma s'optimitza la transmissió. Posarem com a exemple que una xarxa té una MTU de 1500 bytes, la màxima que suporta Ethernet, llavors, el MSS serà 1460. Això s'obté restant als 1500 bytes de MTU, 20 bytes de capçalera IP i 20 de capçalera TCP. Si en un establiment de connexió no es defineix un MSS, llavors s'agafa el que ve definit per defecte, segons el RFC 1122 [5], que és de 536 bytes (MTU de 576 bytes).

La prova es realitza durant l'establiment de connexió, que és on s'ha d'establir el MSS segons el RFC 793 [3]. El que fa la nostra aplicació és enviar un paquet SYN amb la MSS màxima, 1460 bytes, i esperar una resposta. En aquest cas també necessitem poder establir la connexió, per tant és obligatori disposar d'un port obert al destí. Un cop es rep la resposta (si es rep), el que es fa és mirar si conté un MSS, cosa que indicaria que sap llegir i processar la opció, o en canvi, si aquesta resposta no conté la opció, llavors significaria que llegeix el paquet, però no processa la opció. El que també pot passar és que no rebem resposta alguna, llavors la pila destí no suporta opcions TCP.

3.2.4. Distribució de la finestra

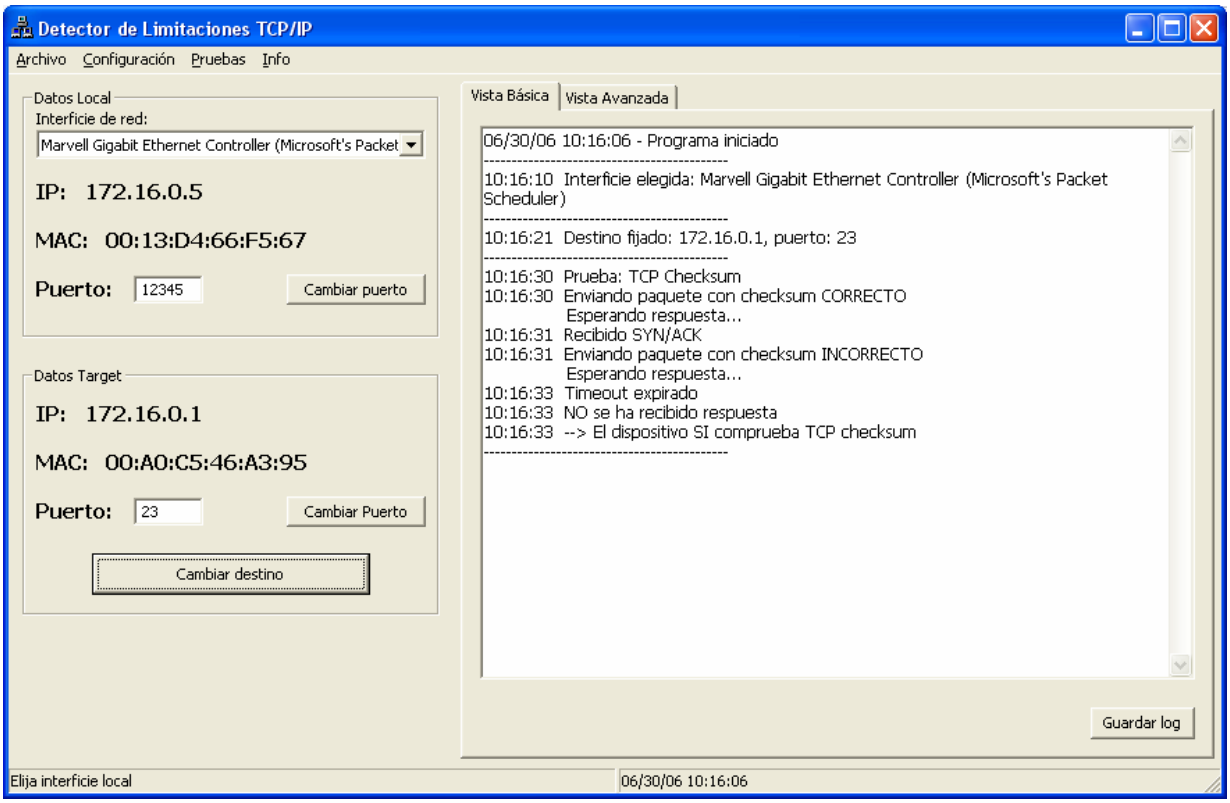

Fig. 3.10 Captura de l'aplicació

Un cop explicats els blocs en els que s'ha distribuït l'aplicació, també cal saber com es troba distribuïda la finestra. El millor és mirar la fig. 3.10, on tenim una

captura de l'aplicació. Es pot veure que el programa consta de tres parts diferenciades:

- *Dades Locals:* Lloc on tenim les dades de la interfície que hem escollit (si la màquina té més d'una interfície, el programa sap diferenciar-les i obtenir les seves dades).

- *Dades Target:* En aquesta segona part tenim les dades de l'equip que volem estudiar. El que es fa és, al començament, demanar a l'usuari la IP del destí, i automàticament s'obté la MAC mitjançant el protocol ARP. El destí es pot canviar sempre que es desitgi.

- *Log:* Aquí es mostren a l'usuari totes les dades que es van obtenint del programa. Hi ha dues vistes del log, una és la bàsica, on es mostren les dades necessàries per a obtenir les conclusions i seguir el procediment de les proves, sense carregar massa la pantalla. La segona vista és l'avançada, on donem a l'usuari les totes les dades disponibles. A més de les dades de la vista bàsica, també es mostra el contingut de les capçaleres dels paquets rebuts, cosa que permet un fer estudi més a fons del dispositiu que estem provant. Aquest log es pot guardar com un arxiu de text.

Les proves es troben a la barra d'eines superior, allí es pot triar la prova a realitzar, sempre que abans es tingui una interfície de sortida seleccionada i un destí fixat.

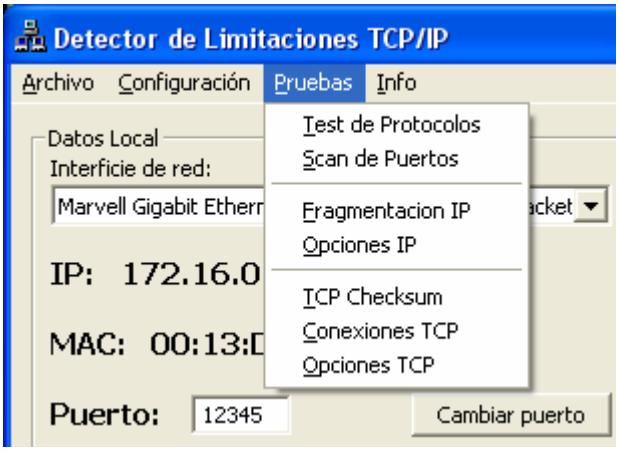

Fig. 3.11 Menú de proves

A la figura 3.11 podem observar el menú de proves desplegat, amb els tests que s'han implementat.

El procediment d'ús del programa, un cop executada l'aplicació seria el següent:

- Escollir interfície de sortida i port
- Escollir IP destí i port

- Realitzar test de protocols per a conèixer quins suporta el destí
- Realitzar les proves que es desitgin.

Aquest procediment no té perquè ser així obligatòriament, però hi ha certs passos que no es poden obviar, com per exemple escollir sortida i destí. La realització del test de protocols no és obligatòria en un principi, però si volem utilitzar una prova que faci ús d'un protocol específic, i aquest no ha estat comprovat, el programa realitza automàticament un test per a saber si la prova es pot fer o no.

CAPÍTOL 4. CONCLUSIONS

En aquest projecte s'ha realitzat estudi sobre les limitacions en la implementació dels protocols IP i TCP, que afecten directament a la pila TCP/IP, i a més, també s'ha desenvolupat una aplicació dedicada a fer els part dels tests que s'han descrit.

Després de fer l'estudi i realitzar proves amb l'aplicació, es va veure que un fet molt important és la predisposició que tingui la màquina a testejar a generar aquestes respostes. Totes les proves que s'han estudiat depenen d'aquestes respostes per a obtenir conclusions i poder deduir els resultats. Part d'aquestes respostes s'obtenen directament a partir de la reacció que té el protocol, però hi ha certes proves que per a obtenir un resultat, s'ha de induir un flux de dades des del destí.

L'aplicació que s'ha implementat conté les proves que va ser possible realitzar mitjançant les reaccions automàtiques que generen els protocols ICMP i TCP, les possibles sense haver d'induir un flux continu, per tant proporciona una base per a la detecció de limitacions sense poder tenir accés al sistema de la màquina destí.

En aquest període de temps, l'aplicació només s'ha pogut provar sobre PCs i sobre algun router, degut a no poder disposar de màquines que implementessin una pila limitada. Aquests dispositius en els que s'ha testejat l'aplicació, tenen implementada la pila en la seva totalitat, per tant superen correctament les proves que s'han realitzat. En un futur, si es poden disposar de dispositius amb piles limitades, seria molt interessant realitzar les proves implementades sobre aquests dispositius i comprovar que els tests determinen les limitacions.

També seria interessant realitzar una ampliació de l'aplicació que s'ha fet, afegint més proves. Es podria implementar un programa que s'executés a la màquina que volem provar, i que en resposta a les ordres que nosaltres li donem, puguem realitzar els tests que no s'han implementat en aquest cas degut a aquest impediment. D'aquesta forma, es podrien crear flux de dades en qualsevol direcció, ja sigui "origen → destí" o "destí → origen", amb la qual cosa, es podria obtenir una descripció molt més detallada de les característiques de la pila.

Les proves estudiades i l'aplicació implementada van destinades a l'entorn que s'ha establert a la introducció. Un dels seus objectius és el d'optimitzar la transmissió de dades per l'enllaç per a que és realitzi de la manera més eficaç possible. Aquesta optimització afecta positivament al disseny dels dispositius, ja que en certa mesura, permet que aquests dispositius tinguin un consum menor. Aquest menor consum implica una reducció de la mida de les bateries que els han d'alimentar, cosa que disminueix la contaminació que aquestes puguin provocar.

BIBLIOGRAFIA

[1] Rodríguez, A. , Gatrell, J. , Karas, J. , Peschke, R. , *TCP/IP Tutorial and Technical Overview*, IBM Redbooks, Agost 2001
http://www.redbooks.ibm.com/abstracts/gg243376.html

[2] “RFC 791: Internet Protocol”, Setembre 1981.
http://www.ietf.org/rfc/rfc791.txt

[3] “RFC 793: Transmission Control Protocol”, Setembre 1981.
http://www.ietf.org/rfc/rfc793.txt

[4] Zaw-Sing Su, “RFC 781: Specification of the IP Timestamp Option”, Maig 1981.
http://www.ietf.org/rfc/rfc0781.txt

[5] Braden, R. , “RFC 1122: Requeriments for Internet Hosts – Communication Layers”, Octubre 1989.
http://www.ietf.org/rfc/rfc1122.txt

[6] Jacobson, V. , Braden, R. , Borman, D. “RFC 1323: TCP Extensions for High Performance”, Maig 1992.
http://www.ietf.org/rfc/rfc1323.txt

[7] Paxson, V. , Allman, M. , Stevens, W. “RFC 2581: TCP Congestion Control”, Abril 1999.
http://www.ietf.org/rfc/rfc1323.txt

[8] Paxson, V. , Allman, M. “RFC 2988: Computing TCP’s Retransmission Timer”, Novembre 2000.
http://www.ietf.org/rfc/rfc1323.txt

[9] “The ISO Model: Theory and Function of Layered Design”, Juliol 2001
http://support.microsoft.com/kb/q103881/

[10] “The uIP Embedded TCP/IP Stack”
http://www.sics.se/~adam/uip/

[11] “lwIP - A Lightweight TCP/IP Stack”
http://savannah.nongnu.org/projects/lwip/

[12] Casals, L. , Paradells, J. , “Internet en las cosas”, Telecom I+D 2005, Novembre 2005, Madrid

[13] Dunkels, A. , “Full TCP/IP for 8-bit Architectures”
http://www.sics.se/~adam/

[14] Microsoft Developer Network Library
Pàgina web de consulta sobre la programació en C++
http://msdn.microsoft.com/library/

[15] WinPcap: The windows packet capture library
Web sobre la llibreria WinPcap, amb tutorials i manual d'us sobre aquesta llibreria
http://www.winpcap.org/

[16] wxWidgets: Cross-Platform GUI Library
Web-site de la llibreria de desenvolupament d'interfícies gràfiques wxWidgets. Conté un complet manual de sobre la llibreria i nombrosos exemples d'aplicació.
http://www.wxwidgets.org/

Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL: Detector de limitacions en piles TCP/IP

AUTOR: Alberto José Juan Torres

DIRECTOR: Lluís Casals Ibáñez

DATA: 7 de juliol de 2006

Annex 1. Passos per crear una aplicació amb WinPcap

En aquest annex s'exposen els passos a seguir per a poder implementar aplicacions que utilitzin la llibreria WinPcap i totes les seves funcions.

- Instalar WinPcap.
- Incloure l'arxiu *pcap.h* al començament de cada arxiu de codi que utilitzi les llibreries.
- Si el programa utilitza funcions específiques de WIN32, s'ha d'incloure *WPCAP* a les definicions del preprocessador.
- Configurar el compilador per a que utilitzi la llibreria *wpcap.lib*. Aquesta llibreria es troba al kit de desenvolupament de WinPcap.
- Configurar el compilador per a que utilitzi la llibreria de WINSOCKS (*winsock32.lib*). És necessària per a algunes de les funcions de WinPcap.

Annex 2. Passos per crear una aplicació amb wxWidgets

Aquest segon annex pretén donar les mateixes referències que l'annex 1, però en aquest cas, per a poder desenvolupar aplicacions amb wxWindows.

- Instalar i compilar les llibreries wxWidgets.
- Incloure l'arxiu *wx/wx.h* al començament de cada arxiu de codi que utilitzi les llibreries.
- Si el programa utilitza funcions específiques de WIN32, s'ha d'incloure *__WXMSW__* , *__WXDEBUG__* i *WXDEBUG=1* a les definicions del preprocessador.
- Configurar el compilador per a que utilitzi les llibreries *wxmsw26d_core.lib wxbase26d.lib wxtiffd.lib wxjpegd.lib wxpngd.lib wxzlibd.lib wxregexd.lib i wxexpatd.lib*. Aquestes llibreries es troben a les carpetes "\lib" , "\contrib\lib" i "\lib\vc_lib", a dins del directori d'instal·lació de wxWidgets.

Una referència per a la configuració específica del compilador Visual Studio de Microsoft per a utilitzar wxWidgets es pot veure aquí:

- http://www.codeproject.com/library/wxwidgets.asp